PhD-FSTC-2011-21
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defense held on 25/11/2011 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Deike PRIEMUTH-SCHMID
Born on 22 May 1981 in Oranienburg (Germany)

ANALYSIS OF RESYNCHRONIZATION MECHANISMS OF STREAM CIPHERS

Dissertation defense committee

Dr. Alex Biryukov, dissertation supervisor
A-Professor, Université du Luxembourg

Dr. Volker Müller, Chairman
A-Professor, Université du Luxembourg

Dr. Frederik Armknecht
A-Professor, Universität Mannheim

Dr. Ralf-Philipp Weinmann, Vice Chairman
Université du Luxembourg

Abstract

Stream ciphers are cryptographic primitives belonging to symmetric key cryptography to ensure data confidentiality of messages sent through an insecure communication channel. This thesis presents attacks on several stream ciphers, especially against their initialization methods.

The first targets are the stream ciphers Salsa20 and Trivium. For both ciphers slid pairs are described. Salsa20 can be distinguished from a random function using only the slid pair relation. When a slid pair is given for Salsa20 both secret keys can be recovered immediately if the nonces and counters are known. Also an efficient search for a hidden slid pair in a large list of ciphertexts is shown. The efficiency of the birthday attack can be increased twice using slid pairs. For the cipher Trivium a large related-key class which produces identical keystreams up to a shift is presented.

Then the resynchronization mechanism of the stream ciphers SNOW 3G and SNOW 2.0 is analyzed. Both ciphers are simplified by replacing all additions modulo 2^{32} with XORs. A known IV key-recovery attack is presented for SNOW $3G^{\oplus}$ and SNOW $2.0^{\oplus}$ where both ciphers have no feedback from the FSM. This attack works for any amount of initialization clocks. Then in both ciphers the feedback from the FSM is restored and the number of 33 initialization clocks is reduced. Chosen IV key-recovery attacks on SNOW $3G^{\oplus}$ with 12 to 16 initialization clocks and SNOW $2.0^{\oplus}$ with 12 to 18 initialization clocks are shown.

In a similar way versions of the stream cipher K2 are attacked. This cipher is simplified by replacing all additions modulo 2^{32} with XORs as well. Chosen IV key-recovery attacks on versions with reduced initialization clocks from five to seven out of 24 are presented. For the version with seven initialization clocks also a chosen IV distinguishing attack is shown.

The last part deals with a linear key-IV setup and known feedback polynomials of the shrinking generator. It is shown that this linear initialization results in a very weak cipher as only a few known IVs are required to recover the secret key. The original design of the shrinking generator does not include any initialization method so the initial state was assumed to be the secret key.

Acknowledgment

Many people have contributed to this thesis. I am grateful to have the opportunity to thank all of them and some people in particular. First and foremost, I wish to thank my supervisor Alex Biryukov for always having time for my problems and for being contactable at almost *any* time. His deep knowledge and encouragement were an ongoing motivation for me through all these years. Special thanks go to my coauthors Bin Zhang and Ralf-Philipp Weinmann for our inspiring discussions on various cryptography topics. In particular I am grateful for Ralf's comments and suggestions and primarily for proof reading my thesis. I would like to thank Volker Müller, Frederik Armknecht and Stefan Lucks for having agreed to be members of my thesis jury. Without Stefan's exciting lectures about cryptography during my studies I would probably never have thought about writing my thesis in this scientific area.

I heartily thank all my colleagues at LACS, especially Dmitry, Alexander, Ivica, Ilja, David, Johann and Jean-François, as well as Barbara, Ton, Sasa and Hugo of the SaToSS group. Lunch breaks would have been much less fun and relaxing without you. I would also like to express my thanks and appreciation to our secretaries Ragga, Mireille and Fabienne for always being supportive with all the administrative matters and for never being too busy for an enjoyable talk.

Additional thanks go to Kay Dittner for proof reading parts of my thesis and correcting my sometimes inscrutable English phrases. As she did not proof read this acknowledgment I hope I found a matching English expression here. And although we have never met I would like to thank Martin Sievers for his helpful advice concerning LaTeX questions.

My boundless gratitude go to my husband Oliver for his help and encouragement through my whole PhD time. While proof reading my thesis he pointed out several incomprehensible sentences and errors. Any errors that still remain are completely my own responsibility. Oliver and our daughter Freya were a continuous impulse to overcome all difficulties to finalize this thesis. They are the best that happened to me and I am looking forward to the years to come.

To my husband Oliver

Contents

Chapter 1

Introduction

1.1 What is Cryptography?

The notation "cryptography" originates from the Greek words $\kappa\rho\upsilon\pi\tau\acute{o}\varsigma$ = kryptós which means secret or hidden and $\gamma\rho\acute{\alpha}\varphi\varepsilon\iota\nu$ = gráphein which means writing. Thus, it is the science of secret transmission, encryption, decryption, code developing, code breaking, etc.

A famous paraphrase of the quintessence of cryptography was given by a well-known cryptographer:

"Cryptography is about communication in the presence of adversaries."

Ron Rivest

This describes in a very clear way one of the main goals of cryptography where two persons want to communicate and hide the shared information from unwanted persons who can listen to their conversation.

The symmetric-key cryptography deals with the protection of communication based on Shannon's model in Figure 1.1.

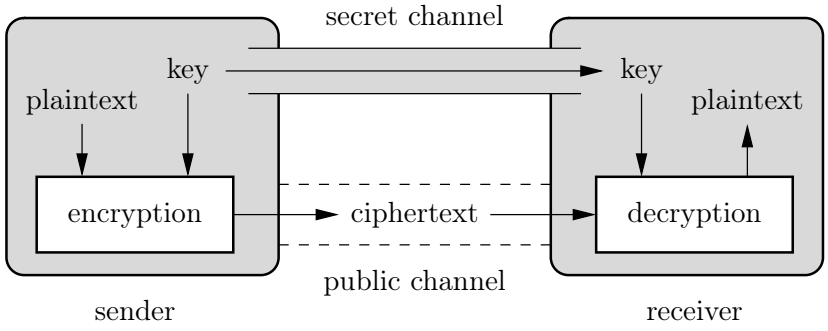

Figure 1.1: Shannon's model

The sender and receiver have access to a secret channel and a public channel. Messages going through the secret channel can not be monitored, altered or erased by an adversary. Though the secret channel is limited in such a way that it is only established at a certain point in time, has a slow transfer rate, can only handle small messages or has other restrictions. The public channel is always available without limitations but messages can be eavesdropped by an intruder. Thus, the sender and receiver have to use a method to encrypt and decrypt their messages. The sender encrypts the message called plaintext to a ciphertext using an encryption algorithm together with a secret key. This key is transmitted via the secret channel and the ciphertext is sent through the public channel. Then the receiver uses the same key to decrypt the ciphertext yielding the plaintext and is able to read the message. The eavesdropper can listen to the ciphertext in the public channel but he can not recover the original message as he does not have the right key.

The symmetric-key encryption algorithms can be divided into stream ciphers, which are the topic of this thesis, and block ciphers. Stream ciphers encrypt the plaintext one digit at a time whereas block ciphers take a "block" of plaintext digits at once and encrypt it.

Another branch of cryptography dealing with the protection of communication is the public-key cryptography. The sender and receiver do not share the same key anymore. Instead the receiver creates a pair of different keys with one key kept secret and the other one broadcasted over the public channel. The so-called public key is used by the sender to encrypt the message which is transmitted over the public channel. Only the receiver knows the secret key connected to the public key and can decrypt the message.

There are many other areas of cryptography such as hash functions, digital signatures, etc., each one invented to achieve another specific goal.

1.2 Security Goals

Over the years the development of cryptography has advanced faster and faster, especially since the invention of computers and even more with the growing importance of inter-computer communication. Not only the methods for cryptography have changed, also the goals for cryptography have altered to meet the requirements of modern communication systems. Nowadays, it is not sufficient anymore to only hide the information of a message from an unwanted person. In modern cryptography four different security goals can be distinguished.

Data confidentiality. This is probably the most widespread goal. Two or more parties want to communicate over an open channel and hide the information from unwanted persons. This can be done by using a cipher

to encrypt the message. The sender encrypts a message with a key so that it usually looks random. Only the receiver who is intended to read the message and therefore has the right key can decrypt the message and get the information. Any person who does not have the key should not be able to get the information from the encrypted message.

User authentication. This goal is also best-known nowadays, since nearly every electronic device which stores and deals with personal information requires a password from the user for the login. The identification of the user to another party which can be another person, a computer or any system with restricted access can be achieved by proving that he knows a secret (password, code), owns a unique object (ID card, token) or has a biometric characteristic (fingerprint, eyes).

Data integrity. With the development of the internet and especially of e-mail communication and file transfer this goal becomes more and more important. When a person sends a message or file to another person then data integrity means that this message is not altered during its way through the communication channel. One way to ensure integrity is to use a Message Authentication Code.

Other goals of cryptography are non-repudiation, access control, anonymity, revocation, certification, timestamping etc.

1.3 Abridgment of Cryptographic History

Although they were probably no form of secret communication, exceptional hieroglyphs chiseled into ancient Egyptian monuments around 1900 BC can be considered the first evidence of classic cryptography [CYC06]. In the sixth century BC Hebrew scholars began to use monoalphabetic substitution ciphers like Atbash or Albam to replace letters based on their order in the alphabet [Coh95]. Though questioned by Kelly [Kel98] the *Scytale* ($\sigma\kappa\upsilon\tau\acute{\alpha}\lambda\eta$) usually is considered a cryptographic tool developed in ancient Greece, supposedly used by Spartan commanders to transmit secret messages. The first transposition cipher was made possible by winding e.g., a strip of parchment around a baton and writing the message onto it (see Figure 1.2). Once unwound the original text was "unreadable" due to the rearranged order of the letters and could be decrypted by using a device of the same diameter.

The probably best-known cipher of ancient history is a substitution cipher known as *Caesar Cipher*. The additive cipher, that replaced each letter by another obtained through shifting in the alphabet, was named after the famous Roman statesman. Concerning the English alphabet each letter is

Figure 1.2: A parchment wound around a Skytale [Wik07]

substituted by the letter with an additional value of an integer constant c modulo 26, in the case of Julius Caesar this value was three.

After several centuries lacking noteworthy advances in both cryptography and cryptanalysis, the first documented description of frequency analysis to break monoalphabetic substitution ciphers was penned by al-Kindi in the ninth century AD [Sin99]. In his "Manuscript for the Deciphering Cryptographic Messages" (*"Risalah fi Istikhraj al-Mu'amma"*) he also provided cryptanalysis techniques for both mono- and polyalphabetic ciphers.

Another important medieval cryptographer was Leon Battista Alberti, the author of a treatise on ciphers in 1467 entitled *"De Cifris"* who was addressed as the "Father of Western Cryptography" by cryptography historian David Kahn [Kah67a]. The Italian invented the *Alberti Cipher Disk* that allowed a polyalphabetic substitution with mixed alphabets and variable period and thus made frequency analysis ineffective due to the masked frequency distribution.

In 1586 the French diplomat Blaise de Vigenère took credit for an idea originally described in the book *"La cifra del Sig. Giovan Battista Bel[l]aso"*. In 1553 Bellaso had introduced a method that used a square table, the so-called *"tabula recta"*[1], and a key termed "countersign" to switch the cipher alphabets for each letter periodically [Kah67a]. The repetition of the usually short key was the starting point for attacks, as guessing the right key length n allowed to split the cipher into n Caesar Ciphers. Nevertheless a version of this method named *Vigenère Cipher* was considered indecipherable until Friedrich Kasiski published a successful general attack on the cipher in his book "Secret Writing and the Art of Deciphering" (*"Die Geheimschriften und die Dechiffri[e]r-Kunst"*) in 1863 [Kas63].[2]

[1] The device is also known as Vigenère square or Vigenère table and can be used for both en- and decryption.

[2] Charles Babbage might have been the first to break the Vigenère Cipher but Kasiski was the first cryptanalyst to publish his results.

The second important publication of the late nineteenth century was Auguste Kerckhoffs's article *"La Cryptographie Militaire"* in 1883 [Ker83] that contained some pioneering ideas whereof some are still fundamental for today's cryptography. A part of this specification known as *Kerckhoffs' principle* will be explained in Section 2.1.

The beginning of the twentieth century brought several innovative concepts to the cryptographic world, in fact one of them actually dated back several years as the basic concept of the *One-Time Pad* originally was delivered in 1882 [Bel11]. The cipher whose perfect secrecy was later proven by Claude E. Shannon is described more detailed in Section 2.1, likewise.

It was World War II to usher in a new era of cryptography. At that time mechanical and electromechanical devices were frequently used for military communication and manual techniques only remained in use where these machines were not suitable. In consequence of these developments the new century was affected by several fundamental changes. Now it were no longer the linguists but the mathematicians working on cryptanalysis and thus trying to develop new methods for encryption, too. And due to the complexity of these new machines the development of elaborate cryptanalysis techniques became inevitable as manually executed brute force attacks were no longer feasible.

The Enigma (from the Greek word $\alpha\acute{\iota}\nu\iota\gamma\mu\alpha$ meaning "riddle") is probably the best-known example for these electromechanical machines. The three-rotor, single-notched Enigma (see Figure 1.3) combined the functionality of a plugboard and three rotating wheels with further setting options to a theoretical amount of 3.3×10^{114} possible configurations which equates 380 bits. Even if the architecture of this far most common model was known, there were still 1.1×10^{23} possible cryptovariables left when trying to determine the daily key, which is comparable to a 76-bit key [Mil01]. Nevertheless, the Enigma Code was cracked due to some of its weaknesses, in particular by exploiting the facts that the plugboard connections were reciprocal[3] and no letter was encrypted to itself [Mah45]. For this purpose the British Government Code and Cypher School had recruited scientists at Bletchley Park, mainly mathematicians like Alan Turing, usually referred to as the "Father of Computer Science". With the deployment of Colossus Mark 1 helping to crack the Enigma Code the digital age had begun in February 1944.

In the late forties Claude E. Shannon published the often-cited article "A Mathematical Theory of Communication" [Sha48] and the fundamental paper "Communication Theory of Secrecy Systems" [Sha49]. The latter contained the already mentioned model of communication and the important proof of the One-Time Pad's perfect secrecy based on results developed during WWII and described in a classified report [Sha45].

[3]Reciprocity implicates that if a certain state would encrypt O to S it would encrypt S to O in the same manner.

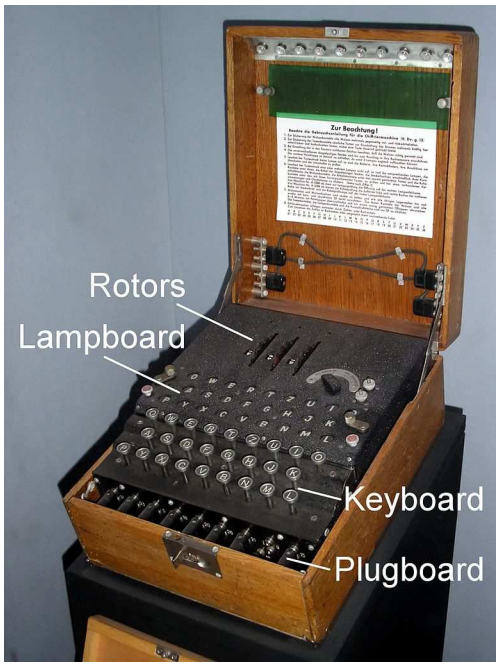

Figure 1.3: A three-wheel Enigma machine [Wik05]

After World War II both cryptography and cryptanalysis was almost exclusively studied by military and intelligence until the scientific community got the chance to participate actively. Two requests for proposals by the National Bureau of Standards in 1973 and 1974 led to the *Data Encryption Standard (DES)* that was developed by researchers of IBM and finally published as Federal Information Processing Standard (FIPS) in 1977. DES and its successors are symmetric-key block ciphers with original DES having only a 56-bit key due to NSA demands[4] contrary to its predecessor *Lucifer* already using 128-bit keys [Joh09].

Another moment of high importance in the seventies is known as *Diffie-Hellman key exchange*, an algorithm that keeps a shared key secret to outsiders although the communication to generate this key can be eavesdropped. Whitfield Diffie and Martin Hellman proposed this innovative method for distributing keys in their article "New Directions in Cryptography" [DH76]. They proposed an asymmetric algorithm combining a so-called "private key" that has to be kept secret with a so-called "public key" that can be made available to anybody and thus especially be communicated via an insecure channel. By computing the shared key based on exponentiation of a generator of a group and the utilization of both private and public key the result

[4]Originally the National Security Agency had even wanted a key length of 48 bits, presumably for "security reasons".

stays exclusive to both participants of the key exchange. Two years later Ron Rivest, Adi Shamir and Leonard Adleman published an article covering the same topic presenting a public key cryptography algorithm known as *RSA* [RSA78] that also was patented in the United States [RSA83].

Although proven insecure being vulnerable to brute force attacks taking 56 hours in 1998 DES was retained as standard until the *Advanced Encryption Standard (AES)* became effective as FIPS in May 2002. Six months earlier the block cipher Rijndael developed by the Belgians Joan Daemen and Vincent Rijmen had been selected as replacement by the National Institute of Standards and Technology[5]. Since then several components developed for AES have been reused in other cryptography algorithms, amongst them the S-box and the MixColumns operation which both will appear in this thesis repeatedly.

1.4 Thesis Outline

Chapter 1 characterizes cryptography in general and gives a description of its goals. Afterwards an abridgement of the history of cryptography is given.

Chapter 2 states the general concept of a stream cipher. It starts with defining a cipher and then introduces the well-known one-time pad cipher which was the motivation for stream ciphers. Afterwards the commonly used building blocks (LFSR, NLFSR, Boolean function, S-box, FSM) are explained.

Chapter 3 contains a survey of cryptanalytic attacks. Before describing several attacks we define the goal of an attack as well as the knowledge of the attacker and which phase (keystream generation or initialization) he strikes.

The first three chapters are influenced to some extent by some books [MvOV96, LN97], PhD theses [Arm06, Hel07, Max06, Ekd03, Lan06, Fis08] and internet sources like Wikipedia [Wik11].

Chapter 4 describes slid pairs for the stream ciphers Salsa20 and Trivium. Both ciphers are in the final portfolio of the eSTREAM project. Given such a slid pair for Salsa20 we can recover both secret keys immediately if the nonces and counters are known. We can also find efficiently a hidden slid pair in a large list of ciphertexts and recover both unknown keys. The slid pairs can also be used to increase the efficiency of the birthday attack twice. For Trivium we show a large related key-class which produces identical keystreams up to a shift. Our results on slid pairs in Salsa20 and Trivium are presented in [PSB08].

Chapter 5 presents attacks against simplified versions of the stream cipher SNOW 3G and its predecessor SNOW 2.0. The two ciphers are simplified by replacing all additions modulo 2^{32} with XORs. For both SNOW 3G and SNOW 2.0 we show a known IV key-recovery attack without feedback from

[5] Until 1988 known as National Bureau of Standards.

the FSM working for any amount of initialization clocks on the one hand and chosen IV key-recovery attacks on reduced numbers of initialization clocks on the other hand. Our results on SNOW 3G are presented in [BPSZ10].

Chapter 6 describes attacks against simplified versions of the stream cipher K2. The simplification is received by replacing all additions modulo 2^{32} with XORs. We present chosen IV key-recovery attacks on versions with reduced initialization clocks from five to seven out of 24 initialization clocks and a chosen IV distinguishing attack on the version with seven initialization clocks. The attack on the reduced version with five initialization clocks and the distinguishing attack will appear in [PS11]. The extension of the chosen IV key-recovery attack to six and seven clocks was obtained later.

Chapter 7 shows that the shrinking generator with a linear key-IV setup and known feedback polynomials can easily be broken using a few known IVs. Originally, the shrinking generator was published without any initialization mode and the initial state was assumed to be the secret key. We studied a linear initialization with key and IV and known feedback polynomials. In this case we only needed a few known IVs to receive the secret key. This work is part of a paper currently under submission.

Chapter 8 gives some final conclusions.

List of publications

During my time as PhD student I have published the following papers which are part of this thesis:

- D. Priemuth-Schmid and A. Biryukov. *Slid Pairs in Salsa20 and Trivium.* In D. R. Chowdhury, V. Rijmen, and A. Das (eds.), INDOCRYPT, volume 5365 of Lecture Notes in Computer Science, pages 1-14. Springer, 2008.

- A. Biryukov, D. Priemuth-Schmid, and B. Zhang. *Analysis of SNOW 3G; Resynchronization Mechanism.* In S. K. Katsikas and P. Samarati (eds.), SECRYPT, pages 327-333. SciTePress, 2010.

- D. Priemuth-Schmid. *Attacks on Simplified Versions of K2.* To appear in P. Bouvry, M. A. Kłopotek, F. Leprevost, M. Marciniak, A. Mykowiecka and H. Rybiński, SIIS, volume 7053 of Lecture Notes in Computer Science, pages 117-127. Springer, 2011.

Currently under submission is the following paper:

- A. Biryukov, R.-P. Weinmann and D. Priemuth-Schmid. *Cryptanalysis of the Shrinking Generator and the Alternating Step Generator using Differentially Related States*

During my PhD time I have also co-authored the following paper which is not included in this thesis:

- A. Biryukov, D. Priemuth-Schmid and B. Zhang. *Multiset Collision Attacks on Reduced-Round SNOW 3G and SNOW 3G*$^{\oplus}$ In J. Zhou and M. Yung (eds.), ACNS, volume 6123 of Lecture Notes in Computer Science, pages 139-153. Springer, 2010.

Chapter 2

Stream Ciphers

Stream ciphers are cryptographic primitives belonging to symmetric key ciphers. A block cipher in a stream cipher mode can simulate a stream cipher so why do we need "real" stream ciphers? In the cryptographic community the opinion prevails that well constructed stream ciphers can have a better performance and / or smaller hardware requirements then block ciphers in a stream cipher mode. The eSTREAM project [eST08] confirms that opinion. eSTREAM was a project by ECRYPT [ECR08] to detect new promising stream ciphers conducted from 2004 to 2008. There are seven stream ciphers in the final portfolio classified into two profiles, a hardware and a software profile. The demand for the hardware profile was that stream ciphers must have less hardware requirements than AES [DR02] whereas the software profile required a better software performance than AES in a stream cipher mode.

2.1 Basic Concept of Stream Ciphers

We start with the definition of a symmetric key cipher.

Definition 2.1: A symmetric key cipher is given by three sets

- a finite set of plaintexts $\mathcal{P}$
- a finite set of ciphertexts $\mathcal{C}$
- a finite set of keys $\mathcal{K}$

and two algorithms

- the encryption algorithm $\mathcal{E}$ with $\mathcal{E} : \mathcal{K} \times \mathcal{P} \rightarrow \mathcal{C}$
- the decryption algorithm $\mathcal{D}$ with $\mathcal{D} : \mathcal{K} \times \mathcal{C} \rightarrow \mathcal{P}$

with the property $\mathcal{D}(k, \mathcal{E}(k, p)) = p$ for each $k \in \mathcal{K}$ and all $p \in \mathcal{P}$.

A fundamental principle which should guide the construction of a cipher is Kerckhoffs' principle [Ker83] which states that a cipher should be secure even if everything except the key is known to the attacker. Thus, the security

of a cipher should not rely on the secrecy of the algorithm rather on a secret key. A key can be kept secure much easier than a whole system, which might be used by several parties and / or implemented on several devices, and a key can easily be changed if it is jeopardized or revealed.

The well-known one-time pad cipher illustrates this principle in an impressive manner since everything except the secret key is known to the attacker. The one-time pad is an advancement of the Vernam cipher [Ver26]. Vernam did not insist on a random key as according to the text of U.S. Patent 1,310,719 [Ver19] he left the choice to the users but suggested the key to be "[...] preferably selected at random [...]". Mauborgne [Kah67b] identified the advantage that an endless and senseless key would make the cryptanalysis more difficult. The one-time pad is a Vernam cipher with completely random keys which are used only once.

Definition 2.2: The one-time pad is a cipher with the following properties:

- the key is completely random
- the key is at least as long as the plaintext
- each key is used only once for encryption.

As encryption function the XOR or a modulo addition can be used. Table 2.1 gives an example of the encryption with the one-time pad. A number from 0 to 25 is dedicated each letter of the alphabet, e.g., a=0, b=1, ..., z=25. Then the encryption function is the addition modulo 26.

Plaintext:

t	h	i	s	i	s	m	y	t	e	x	t	t	o	e	n	c	r	y	p	t
19	7	8	18	8	18	12	24	19	4	23	19	19	14	4	13	2	17	24	15	19

Encryption with random key:

	h	k	o	r	o	m	g	i	g	c	w	b	z	f	l	x	w	v	q	m	y
+	7	10	14	17	14	12	6	8	6	2	22	1	25	5	11	23	22	21	16	12	24

Ciphertext:

	a	r	w	j	w	e	s	g	z	g	t	u	s	t	p	k	y	m	o	b	r
=	0	17	22	9	22	4	18	6	25	6	19	20	18	19	15	10	24	12	14	1	17

Table 2.1: Example for encryption with the one-time pad

In 2011 Bellovin [Bel11] discovered that Frank Miller invented the one-time pad 35 years earlier. The one-time pad was proven to be unconditionally secure by Shannon [Sha49]. *Unconditionally secure* means that even an attacker with unlimited resources (computing power, time, memory requirements, etc.) is unable to recover the unknown plaintext or the secret key from a ciphertext. This advantage of a security proof is confronted with the fact that this cipher is infeasible for most practical applications. The constant need of truly random keys is a problem as the generation of random

keys is extensive due to a required source of true randomness (for example radioactive decay). Also the safe storage or transmission of the secret keys which are at least as long as the plaintexts is difficult.

This is the motivation for stream ciphers which take a short secret key and produce long pseudo-random sequences, so-called keystreams, to encrypt the plaintexts. Thus, stream ciphers only imitate the one-time pad, the unconditional security does not hold anymore. Instead, the security of stream ciphers is described as *conditionally secure* which means that in principle the cipher can be broken but the attacker would need an unrealistic amount of resources (computing power, time or memory requirements, etc.).

The simplest way to break a stream cipher is the brute force attack where the attacker exhaustively tries all keys until he has found the right one. The complexity of this exhaustive key search is based on the amount of possible keys. With an increasing amount of potential keys the complexity grows exponentially. Therefore, the quantity of possible keys should be big enough to make the brute force attack infeasible in practice. Consequently, a stream cipher is called secure if no attack exists with a smaller complexity than the brute force attack.

The algorithm to compute the keystream is called a keystream generator.

Definition 2.3: A KeyStream Generator (KSG) is a Finite State Machine (FSM) which consists of

- a finite set of keys $\mathcal{K}$
- a finite set of keystreams $\mathcal{Z}$
- an internal state $\mathcal{S}$
- an update function $g(\,)$
- an output function $f(\,)$.

First the FSM is loaded with the key. Then for each clock the update function updates the internal state and the output function produces the element of the keystream.

Definition 2.4: A stream cipher is a symmetric key cipher which encrypts the elements of the plaintext one at a time. It consists of a KSG which generates the keystream and a combining function which combines the keystream and the plaintext elements.

The combining function is usually an XOR but also the modulo addition is used. Stream ciphers are typically classified into *synchronous* and *self-synchronizing* ones but there are also stream ciphers which do not belong to any of these categories, for example Phelix [WSLM05] which has a built-in message authentication code functionality.

Definition 2.5: A *synchronous* stream cipher generates the keystream independently from the previous ciphertext elements.

The notation "synchronous" means that the sender and receiver must be synchronized for a successful decryption. If a ciphertext element is deleted or inserted during its way through the communication channel then synchronization is lost and all the following ciphertext elements can not be decrypted correctly. If a ciphertext element is modified only the corresponding plaintext element is changed and no other plaintext elements are affected. Such a modification of one element might not be detected. Message authentication is needed to prevent both incidents.

Definition 2.6: In a *self-synchronizing (asynchronous)* cipher a fixed number of previous ciphertext elements are involved in the keystream generation.

The notation "self-synchronizing" means that the cipher will synchronize itself after a fixed number of ciphertexts if an error occurs.

Stream ciphers must use a new random key for each new plaintext. As described, the generation of random keys is expensive so the designers looked for a possibility to use one key for more than one plaintext. The solution is the use of an initial value (IV) in addition to the key. Nearly all new cipher constructions use a key together with an IV. This IV is typically publicly known and should be a random nonce (number used once) or a counter. The combination of the same key and IV together must never be used again. An initialization phase is needed to have a proper mixing of the unknown key and the known IV before the keystream is produced. This initialization phase should be non-linear, unequal if procurable to the update function used during the keystream generation as well as long enough to counteract attacks against the initialization method. Figure 2.1 shows a synchronous stream cipher which takes a key and an IV to produce the keystream.

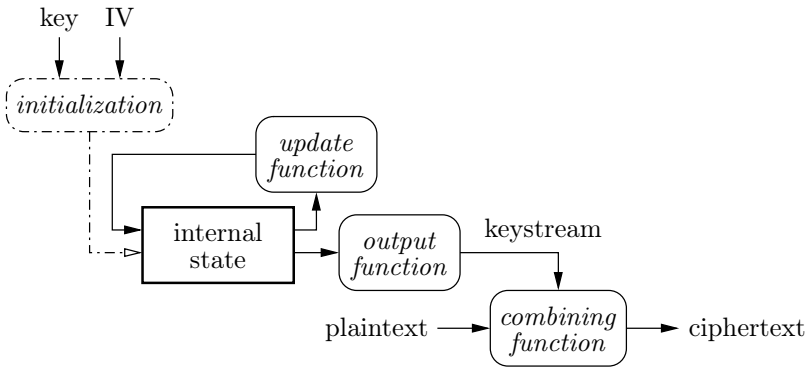

Figure 2.1: Synchronous stream cipher with key and IV

2.2 Building Blocks

Stream ciphers can be constructed in many different ways. We describe the widely used building blocks. There are several other constructions.

2.2.1 Linear Feedback Shift Register

One of the most frequently used building block of stream ciphers is the Linear Feedback Shift Register (LFSR). It is a FSM operating on some finite field $\mathbb{F}_q$. This finite field is often a binary field with $q = 2$ or an extension of the binary field with $q = 2^e$ where $1 < e \in \mathbb{N}$. The register consists of a number of cells which gives the length of the register denoted with l. Each cell can contain one element of $\mathbb{F}_q$. The content of the register is called the *state* and the content at the beginning of the computation is called *initial state* or *starting state*. Figure 2.2 shows a picture of an LFSR.

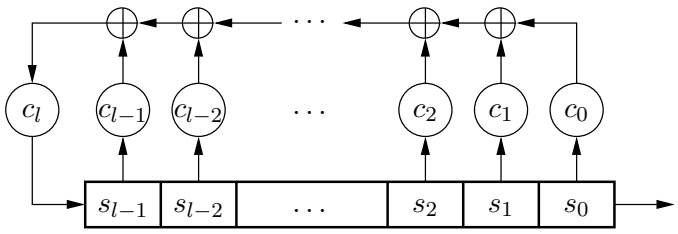

Figure 2.2: General composition of a Linear Feedback Shift Register

When the LFSR is clocked (the time is denoted with t) the content of the last (rightmost) cell becomes the output, for all other cells the content is shifted one cell to the right and the content of the first (leftmost) cell is computed via the *linear recurrence relation*

$$s_{t+l} = c_l \cdot \sum_{i=0}^{l-1} c_i \cdot s_{t+i} \qquad c_0, c_1, \ldots c_l \in \mathbb{F}_q$$

where all computations are done in $\mathbb{F}_q$. The constants $c_0, c_1, \ldots, c_l$ are the *feedback coefficients* with $c_0 \neq 0$ and $c_l \neq 0$. The so-called *tap positions* are the connections at $c_i \neq 0$ for $i = 0, \ldots, l-1$ and the connection at c_l is called the *feedback position*.

Definition 2.7: The *feedback polynomial* of an LFSR is the polynomial

$$g(x) = c_l x^0 - c_{l-1} x^1 - \cdots - c_0 x^l$$

in the ring of polynomials $\mathbb{F}_q[x]$.

There are some important properties which the feedback polynomial must fulfil in order to achieve a long period.

Definition 2.8: A polynomial $g(x) \in \mathbb{F}_q[x]$ with degree l is called *irreducible* if it can not be factored into two polynomials with positive degrees smaller than l in $\mathbb{F}_q$.

Definition 2.9: An irreducible polynomial $g(x) \in \mathbb{F}_q[x]$ with degree l is called *primitive* if it divides the polynomial $x^{q^l-1} - 1$ but no polynomial $x^u - 1$ with $u < q^l - 1$.

Theorem 2.10: If an LFSR has a primitive feedback polynomial $g(x) \in \mathbb{F}_q[x]$ with degree l, then the period of the LFSR is $T = q^l - 1$ for every non-zero starting state.

For the proof we refer to [LN97].

Therefore, an LFSR with a primitive feedback polynomial is called a *maximum length LFSR* and its sequence is called *maximum length sequence*.

Definition 2.11: The *linear complexity* of a sequence $\mathbf{s} = s_0, s_1, \ldots, s_i \in \mathbb{F}_q$, denoted with $LC(\mathbf{s})$, is the length of the shortest LFSR that can generate this sequence.

If the sequence $\mathbf{s}$ is the all-zero sequence then $LC(\mathbf{s}) = 0$. If the sequence $\mathbf{s}$ is truly random then $LC(\mathbf{s}) = \infty$ because no LFSR can produce such a sequence.

The Berlekamp-Massey algorithm [Mas69] can efficiently find the shortest LFSR for a sequence $\mathbf{s}$ and hence its linear complexity if at least $2\,LC(\mathbf{s})$ elements of $\mathbf{s}$ are given.

2.2.2 Non-Linear Feedback Shift Register

One solution to solve the linearity problem of an LFSR is the usage of a Non-Linear Feedback Shift Register (NLFSR). It is built in the same way as an LFSR but the feedback function is non-linear. Thus, it preserves the advantage to be easily implemented. In addition, NLFSRs have high linear complexity which makes the execution of algebraic attacks much harder or infeasible. One disadvantage is that balancedness is hard to achieve. Another problem is the little knowledge of the mathematical background of NLFSRs contrary to LFSRs. Thus, no general criteria for maximum periods or good statistical properties are known.

2.2.3 Boolean Function

Definition 2.12: A Boolean function is a mapping $f : \mathbb{F}_2^n \to \mathbb{F}_2$

$$f(x_1, x_2, \ldots, x_n) = y \quad \text{with} \quad x_1, x_2, \ldots, x_n, y \in \mathbb{F}_2.$$

There are 2^{2^n} different Boolean functions with n input variables. We denote this set of Boolean functions with $\mathcal{B}_n$. Boolean functions can be represented in different ways.

Definition 2.13: The *truth table* (TT) of a Boolean function $f \in \mathcal{B}_n$ is a binary string of length 2^n

$$TT(f) = [f(0,0,\ldots,0), f(1,0,\ldots,0), f(0,1,\ldots,0), \ldots, f(1,1,\ldots,1)] \ .$$

The truth table is the easiest way if the number n of inputs is not too big.

Definition 2.14: The *algebraic normal form* (ANF) of a Boolean function $f \in \mathcal{B}_n$ is a polynomial over $\mathbb{F}_2$

$$f(x_1, x_2, \ldots, x_n) = c_0 \oplus \bigoplus_{1 \le i \le n} c_i x_i \oplus \bigoplus_{1 \le i < j \le n} c_{ij} x_i x_j \oplus \cdots \oplus c_{12\ldots n} x_1 x_2 \ldots x_n$$

with the coefficients $c_0, \ldots, c_n, c_{12}, \ldots, c_{12\ldots n} \in \mathbb{F}_2$.

This representation as a polynomial is unique.

Definition 2.15: Let $f \in \mathcal{B}_n$ and $x, w \in \mathbb{F}_2$ with $x = (x_1, x_2, \ldots, x_n)$ and $w = (w_1, w_2, \ldots, w_n)$ and $x \cdot w = x_1 w_1 \oplus x_2 w_2 \oplus \cdots \oplus x_n w_n$. Then the *Walsh transform* of $f(x)$ is a real valued function over $\mathbb{F}_2^n$ defined as

$$W_f(w) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x) \oplus x \cdot w} \ .$$

The *Walsh spectrum* is the integer valued vector W_f.

For cryptographic use some properties like algebraic degree, balancedness, non-linearity, correlation immunity, algebraic immunity are important.

Definition 2.16: The *algebraic degree* of $f \in \mathcal{B}_n$ denoted as $\deg(f)$ is the number of variables in the highest order term with a non-zero coefficient in the ANF.

Definition 2.17: A function $f \in \mathcal{B}_n$ is called *affine* if it has no term with degree > 1 in the ANF. The set of all affine functions with n input variables is denoted as $\mathcal{A}_n$. A function $f \in \mathcal{A}_n$ is called *linear* if it has a constant term equal to zero.

For some properties we need the Hamming weight and the Hamming distance so we define them as well.

Definition 2.18: The *Hamming weight* of a binary sequence $\mathbf{s}$ denoted as $w_H(\mathbf{s})$ is the number of ones in $\mathbf{s}$. For two binary sequences $\mathbf{s}_1$ and $\mathbf{s}_2$ with the same length the *Hamming distance* denoted as $d_H(\mathbf{s}_1, \mathbf{s}_2)$ is the number of positions where both sequences differ. Thus, the equation $d_H(\mathbf{s}_1, \mathbf{s}_2) = w_H(\mathbf{s}_1 \oplus \mathbf{s}_2)$ holds.

Definition 2.19: A function $f \in \mathcal{B}_n$ is called *balanced* if its truth table has an equal number of ones and zeros, i.e.,

$$w_H(TT(f)) = 2^{n-1} \quad \Leftrightarrow \quad W_f(0) = 0 \ .$$

Definition 2.20: The *non-linearity* of $f \in \mathcal{B}_n$ denoted as $\mathrm{nl}(f)$ is the minimum distance to the set $\mathcal{A}_n$, i.e.,

$$\mathrm{nl}(f) = \min_{g \in \mathcal{A}_n} (d_H(f, g)) \quad \Leftrightarrow \quad \mathrm{nl}(f) = 2^{n-1} - \frac{1}{2} \max_{w \in \mathbb{F}_2^n} |W_f(w)| \ .$$

Definition 2.21: A function $f \in \mathcal{B}_n$ is called m^{th} *order correlation immune (CI)* if and only if its Walsh transform satisfies

$$W_f(w) = 0 \quad \forall \ w \text{ with } 1 \leq w_H(w) \leq m \ .$$

If f is also balanced then it is called m*-resilient*.

Definition 2.22: The *algebraic immunity* for a function $f \in \mathcal{B}_n$ denoted as $AI(f)$ is the minimum degree of $g \in \mathcal{B}_n$ with $f \cdot g = 0$ or $(f \oplus 1) \cdot g = 0$.

2.2.4 S-box

A Substitution Box (S-box) is a mapping $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$. It substitutes a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{F}_2^n$ by a vector $\mathbf{y} = (y_1, y_2, \ldots, y_n) \in \mathbb{F}_2^m$ in a non-linear manner. The mapping f can be divided into m Boolean functions. Then we can write $\mathbf{y} = f(\mathbf{x})$ as

$$\begin{aligned} y_1 &= f_1(x_1, x_2, \ldots, x_n), \\ y_2 &= f_2(x_1, x_2, \ldots, x_n), \\ &\vdots \\ y_m &= f_m(x_1, x_2, \ldots, x_n). \end{aligned}$$

Therefore, the S-box can be defined by the truth tables or the ANF of its Boolean functions. The Boolean functions are also used to define some properties of the S-box.

Definition 2.23: An S-box $f = (f_1, f_2, \ldots, f_m)$ is *balanced* if all non-zero linear combinations of its Boolean functions are balanced.

Definition 2.24: The *algebraic degree* of an S-box f is defined as the minimum algebraic degree of all non-zero linear combinations of its Boolean functions, i.e.

$$\deg(f) = \min_{(c_1,c_2,\ldots,c_m) \in \mathbb{F}_2^m \setminus \{0\}} \deg\left(\sum_{i=1}^{m} c_i f_i\right) \quad .$$

Definition 2.25: The *non-linearity* of an S-box f is defined as minimum non-linearity of all non-zero linear combinations of its Boolean functions, i.e.

$$\mathrm{nl}(f) = \min_{(c_1,c_2,\ldots,c_m) \in \mathbb{F}_2^m \setminus \{0\}} \mathrm{nl}\left(\sum_{i=1}^{m} c_i f_i\right) \quad .$$

There are two kinds of S-boxes, static ones and varying ones. The static S-box is just a fixed mapping which can be designed to have special properties and which is assumed to be known to the attacker. The varying S-box is key dependent and thus varies during the computation.

The design of good S-boxes is a difficult task. This might be the reason why a good S-box is often reused by designers of other ciphers, for example the 8-bit to 8-bit S-box of the AES block cipher [DR02]. The input is mapped to its multiplicative inverse in the finite field, followed by an affine transformation. Amongst others, the stream ciphers SNOW 2.0 [EJ02], SNOW 3G [ETS06a] and K2 [KTS07] use this S-box.

2.2.5 Finite State Machine

The main intention of using a Finite State Machine (FSM) is to increase non-linearity in a stream cipher construction. Usually an FSM is used together with one or more LFSRs. It consists of some extra memory and a non-linear update function. A non-linear operation is mostly more expensive then a linear operation. Thus, stream ciphers have to find a tradeoff between a large state which can be updated fast and easily and non-linearity. The FSM is one solution for this problem. A typical construction of an FSM has some memory cells which are updated by S-boxes and one or more inputs from the LSFR(s). The output produced by the FSM is combined with the output of the LFSR(s) to yield the keystream. During the initialization phase the output of the FSM is often fed back in the LFSR(s) to achieve a non-linear initialization phase and a non-linear state of the LFSR(s).

2.2.6 Additional Components

The previously described building blocks need to be connected to result in a stream cipher construction. Nowadays, usually non-linear functions are used to combine several building blocks and to compute the keystream.

The non-linear combining function uses the output of several (N)LFSRs to compute the output of the stream cipher in a non-linear way, whereas the non-linear filter function uses several elements of only one (N)LFSR to compute the output of the stream cipher in a non-linear way. Figure 2.3 shows both functions which are usually Boolean functions if $\mathbb{F}_q = \mathbb{F}_2$.

The irregular clocking is another method used to increase the non-linearity. There are different kinds of irregular clocking. One uses the outputs or some elements of one or more (N)LFSRs to determine when and how often which of the (N)LFSRs are clocked. An example is the A5/1 algorithm [BGW99] where for each cycle two or three out of its three registers are clocked. Another way to achieve an irregular clocking is the use of regular clocked (N)LFSR(s) but decimate the output yielding a keystream with unknown clocking decisions. Two examples for such a decimation are the shrinking generator [CKM93] and the self-shrinking generator [MS94].

This enumeration is far from being complete as there are many more elements and / or methods to build a stream cipher.

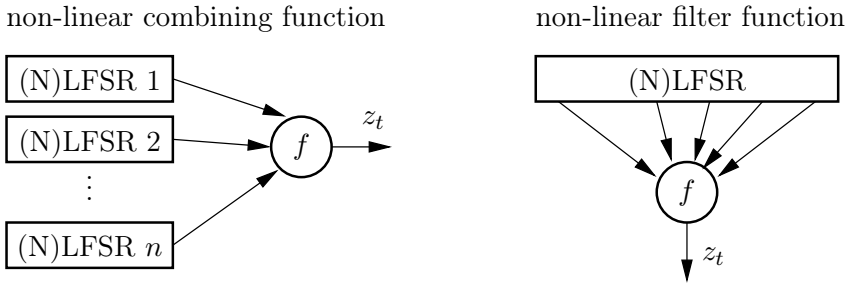

Figure 2.3: Two functions to increase non-linearity

2.3 Stream Ciphers Analyzed in this Thesis

All ciphers analyzed in this thesis are synchronous stream ciphers which use a key together with an IV except for the last cipher.

Our first topic is Salsa20 [Ber05], a stream cipher inspired by a block cipher in counter mode. It consists of a 4×4 - matrix of 32-bit words together with a non-linear update function. This cipher has no initialization phase. The keystream is produced by loading the key, a nonce and a counter in the matrix and performing the update function several times. As long as keystream is needed this procedure is repeated with an increased counter.

The second stream cipher named Trivium [dCP05] is a simple but elegant design of one NLFSR over $\mathbb{F}_2$ with three non-linear update functions having three different feedback positions and a linear filter function for the

keystream computation. The update of the internal state during the initialization phase of $4 \cdot 288$ clocks is the same as during the keystream generation. The ciphers Salsa20 and Trivium are analyzed in Chapter 4.

Both SNOW [EJ02, ETS06a] ciphers use an LFSR with sixteen 32-bit words, an FSM containing either three memory words connected with two S-boxes or two memory words connected with one S-box and a non-linear keystream function. After loading the key and IV in the LFSR both ciphers perform an initialization phase of 32 clocks where the output of the FSM is fed back in the LFSR resulting in a non-linear initialization phase. Then both ciphers switch to the keystream generation phase but the first keystream word is discarded. Key-recovery attacks on simplified versions with and without feedback from the FSM in the LFSR during the initialization are presented in Chapter 5.

The next cipher named K2 [KTS07] is special as it uses a dynamic feedback controller which dynamically chooses between four different linear feedback functions for one FSR consisting of eleven 32-bit words. The other building blocks are another FSR of five 32-bit words, an FSM with four memory words and four S-boxes as well as a non-linear keystream function combining words from all three building blocks to produce two keystream words each clock. During the initialization phase the two keystream words are fed back in the FSRs one in each. Key-recovery attacks as well as a distinguishing attack on simplified versions are shown in Chapter 6.

The last one is the shrinking generator [CKM93], an old design which uses two LFSRs over $\mathbb{F}_2$ where the output bit of the first LFSR decides whether the output bit of the second LFSR becomes the keystream bit or is discarded. This construction is given without any key-IV loading or initialization method. Thus the initial states are assumed to be the secret key which sometimes includes the feedback polynomials of the LFSRs, too. A version with a linear key-IV setup and known feedback polynomials is studied in Chapter 7.

In this chapter we gave some basics about stream ciphers. We introduced the one-time pad and its relation to stream ciphers. This is followed by a short overview of the most used building blocks. We also gave a classification of the stream ciphers analyzed in this thesis. In the next chapter we will give a survey of attacks applicable to stream ciphers.

Chapter 3

Survey of Attacks

Prior to the description of different attacks on stream ciphers we have to define the goal of an attack, the knowledge of the attacker and which phase (initialization or keystream generation) the attacker strikes.

There are two main goals in attacking a stream cipher. The more important one is to recover the secret key. Consequently, an attack with this goal is called a *key recovery attack*. The other goal is to distinguish the keystream computed by the stream cipher from a truly random stream. These attacks are called *distinguishing attacks*. A key recovery attack is much more dangerous as the attacker is able to decrypt all ciphertexts which were encrypted with this key. If an attacker can recover the secret key he can also distinguish the produced keystream from a random stream. There are also other goals possible, for example *state recovery*.

The knowledge of an attacker can be divided into four categories. In a *ciphertext only attack* he knows only the ciphertext. The *known plaintext attack* gives the attacker access to known plaintext as well as the corresponding ciphertext. In the stream cipher case this means that the attacker knows the keystream. In a *chosen plaintext attack* the attacker is allowed to choose the plaintexts and get the ciphertexts. The other way around works the *chosen ciphertext attack* where the attacker chooses the ciphertexts and gets the decrypted plaintexts. It is also possible that the attacker has additional knowledge, for example that two keys are related somehow.

As most stream ciphers execute an initialization phase before they produce the keystream the attacker can decide which of these two phases he likes to strike. The attacks on the initialization phase can be divided into *known IV attacks* and *chosen IV attacks* depending on whether the attacker only knows the IVs or is able to choose them. An attack on the keystream phase usually assumes a known-keystream scenario (i.e. known plaintext attack).

3.1 Brute Force Attack

This kind of attack is always possible as the attacker can always try all potential keys exhaustively. He does not exploit any weakness the stream cipher might have. As mentioned in Section 2.1 the complexity is based on the amount of possible keys and grows exponentially with an increasing amount of possible keys. If keys of size n bits are used then the amount of possible keys is 2^n which leads to 2^n attempts to find the right key. The complexity of the brute force attack is only an upper bound because the search for the right key can be parallelized.

3.2 Time/Memory Tradeoffs

Time/memory tradeoff attacks on stream ciphers have been independently described by Babbage [Bab95] and Golic [Gol97] adapting Hellman's preceding work on block ciphers [Hel80]. Their results were improved by Biryukov and Shamir [BS00].

Usually, the attacks are based on the findings of the birthday paradox stating the astonishingly high probability of two persons of a relatively small group having the same birthday – in a group of 23 people the probability is 50.7 % that this phenomenon occurs. The time/memory tradeoff attack consists of two phases, a preprocessing phase and a real-time phase. During the first phase the general structure of the cipher is analyzed and the results are collected in large tables. In the second phase the preparatory work is used in connection with data produced from a particular unknown key to recover it. Thus, the attack can be regarded as a brute force attack on a smaller key space.

3.3 Correlation Attacks

An important kind of attacks are correlation attacks. They were introduced by Siegenthaler [Sie84, Sie85] and target stream ciphers which combine several LFSRs with a non-linear Boolean function to compute the keystream bit. These attacks use linear correlations between the keystream bits and the output bits of one or a small number of LFSRs to recover the secret key. If the stream cipher consists of n LFSRs each having length l_i with $i = 1, \ldots n$ then the brute force attack would need $\prod_{i=1}^{n} 2^{l_i}$ tries to recover the initial states of the LFSRs and thus the secret key. With correlations between the keystream and single LFSRs the initial states of the LFSRs can be recovered one by one which would need only $\sum_{i=1}^{n} 2^{l_i}$ tries which is a huge step down. The correlation immunity of a Boolean function can be computed via the Walsh transform. If a function is m^{th} order correlation immune then $m + 1$ LFSRs must be considered together.

The approach to approximate the keystream by a linear combination of the output of some LFSRs is related to coding theory. The keystream is considered as a polluted linear combination of the output bits of the LFSRs. Therefore, methods from coding theory have been used to improve correlation attacks [MG90, Pen96].

Fast correlation attacks [MS89, CS91, JJ99, JJ00] not only use weaknesses in the Boolean functions but also exploit the linearity of the LFSRs' update functions. These attacks search during a preprocessing step for parity check equations and use them in the processing step to recover the initial state of the LFSRs.

3.4 Differential Cryptanalysis

Biham and Shamir [BS90] introduced the differential cryptanalysis in 1990. The main target for differential attacks are block ciphers but this method can also be used to attack hash functions and stream ciphers. The idea is to find differences in the plaintext so that the differences in the ciphertext have some recognizable characteristics. For stream ciphers the differences are placed in the IV and the keystream is checked for special differences. Depending on the capabilities of the attacker we can talk about known or chosen IV attacks. The goal of differential attacks is either to distinguish the keystream from a random function or to recover the key. Typically the differences are computed via an XOR. The differences in the IVs are traced through the clocks of the initialization phase. For all linear building blocks the differences can be calculated directly. For the non-linear parts the probability that a given input difference leads to a special output difference must be computed. In the case of an S-box the behavior of the differences can be represented by its *difference distribution table*. This table of size $2^n \times 2^m$ contains as rows the input differences and as columns the output differences. At the intersection of a row and a column is a number which shows how often this input difference results in this output difference. The differential attacks on stream ciphers have been studied in [ALP04].

3.5 Algebraic Attacks

The general idea for algebraic attacks is to model the known keystream as a system of equations in the unknown key variables and then solve this system to derive the secret key. The idea for such a method was brought up by Shannon [Sha49].

A system of linear equations can be solved quite easily using Gaussian elimination whereas the general solution of a system of non-linear equations over a finite field is an NP-complete problem.

There are several methods to solve systems of non-linear equations. The easiest one to solve a system of non-linear equations is to replace each monomial with a degree higher than one by a new variable which is treated independently from the already existing ones. The result is a system where all equations are linear but the number of variables has increased considerably. So the bottleneck is to get enough linear independent equations that Gaussian elimination is applicable, more precisely the number of equations should be greater or equal to the number of variables.

Another widely-used technique is Buchberger's algorithm [Buc65] which computes Gröbner bases to solve a system of non-linear equations. The theory of Gröbner bases is also used by Faugère who developed the algorithms F_4 [Fau99] and F_5 [Fau02]. These algorithms are the most efficient methods known so far to solve non-linear equations over a finite field. The drawback for the algorithms based on Gröbner bases is the difficult estimation of the complexity.

An algorithm to solve an overdefined system of quadratic equations over $\mathbb{F}_2$ was presented by Kipnis and Shamir [KS99]. This relinearization algorithm was analyzed and strengthen by Courtois et al. [CKPS00] to handle higher degree multivariate systems. This improved version was called XL algorithm where XL comes from eXtended Linearization was used to attack the stream cipher Toyocrypt [Cou02]. The criteria for success and complexity of the XL algorithm are unsettled as well as for another improvement the XSL algorithm [CP02].

In 2003 two papers [CM03, AK03] started a surge on research on algebraic attacks. One year later, Meier et al. [MPC04] introduced the *algebraic immunity* of a Boolean function which was a generalization of [CM03].

Definition 3.1: The *algebraic immunity* of a Boolean function f, denoted with $AI(f)$, is the smallest degree of a Boolean function g which satisfies

$$g \cdot f = 0 \text{ or } g \cdot (f \oplus 1) = 0 \ .$$

Then the function g is called an annihilator for f.

An algorithm to find the algebraic immunity for a Boolean function f with n variables is also presented in [MPC04]. The algorithm runs with a complexity of $O((\frac{n}{d})^3)$ with d being the algebraic immunity. Two years later Armknecht et al. [ACG+06] gave an improved algorithm with complexity $O((\frac{n}{d})^2)$. A lot of research concerning algebraic attacks and algebraic immunity followed (e.g., [BL05, DGM05, Car06, FM07]).

Fast algebraic attacks are an extension of algebraic attacks and were introduced by Courtois [Cou03] and improved by Armknecht [Arm04]. The idea is to reduce the degree of the system of equations in a precomputation step by eliminating all high degree monomials which are independent of the keystream bits.

Carlet and Feng [CF08] showed an infinitive class of Boolean functions with good properties regarding balancedness, non-linearity, algebraic immunity and immunity against fast algebraic attacks.

3.6 Cube Attacks

A very new kind of attack are cube attacks introduced in 2009 by Dinur and Shamir [DS09]. They consider the stream cipher as a tweakable polynomial in the public variables coming from the IV and the secret variables coming from the key. The attacker is allowed to tweak the polynomial by choosing different values for the public variables. During a precomputation step several systems of equations with low degree are constructed by varying the public variables for all possible values. Then these systems are solved to get the solution in the secret variables following the secret key. Cube attacks are the currently best known attack against Trivium [DS09] and Grain-128 [DS11].

3.7 Side-Channel Attacks

Instead of looking for weaknesses of algorithms with cryptanalysis, side-channel attacks exploit leaked information gained from the physical implementation of cryptographic devices. Side-channel attack can be divided into two categories *passive attacks*, which only monitor the behavior of the devices performing a cryptographic computation, and *active attacks*, which try to influence the device during the computation by physical means to cause errors or irregular behavior. Active attacks are usually called *fault attacks*. Passive attacks can observe different sources of side-channel information.

Timing attacks are based on data dependent differences in execution time measured while a computation occurs. Kocher [Koc96] explained how to analyze timing information on various public key algorithms, amongst others Diffie-Hellman [DH76] and RSA [RSA78]. Differences in processing zeros and ones permit correlation analyses and hence the discovery of the secret key.

Power monitoring attacks or *power analyses* utilize slight variations in power consumption of the operating hardware for the retrieval of information. Kocher et al. [KJJ99] describe two approaches. In a simple power analysis (SPA) the power trace of a cryptographic device is observed over time, allowing to identify specific mathematical operations and even the key. A differential power analysis (DPA) uses statistical methods for the interpretation of these measurement results. By involving signal processing and error correction functions e.g., to filter out noise, relevant information can be extracted even better than with SPA.

Electromagnetic attacks take advantage of electromagnetic waves that are emanated by electronic devices and hence are comparable to the power analysis. When the methods described above are adapted to the exploitation of electromagnetic radiation, they are called SEMA or DEMA [Rec04].

There also exist further forms of attacks like *acoustic attacks* that make use of the sounds emitted e.g., by CPUs or *thermal imaging attacks* based on infrared images of a processor.

The countermeasures for side-channel attacks are as diverse as the attacks themselves. To prevent timing attacks, the duration of data dependent operations can be changed e.g., by equalization of computation time depending on the data to process or inserting random delays. In general, the masking or blinding of data is suitable to counteract multiple forms of side-channel attacks. To avoid these attacks the conditional execution paths for the processed data must not depend on secret but on public information.

In this chapter we gave a survey of existing attacks against stream ciphers. This list is far from being complete several other techniques exist.

Chapter 4

Slid Pairs in Salsa20 and Trivium

In 2005 Bernstein [Ber05] submitted the stream cipher Salsa20 to the eSTREAM project [eST08]. Original Salsa20 has 20 rounds, later 8 and 12 rounds versions Salsa20/8 and Salsa20/12 were also proposed. Now Salsa20/12 is in the software profile in the eSTREAM portfolio. It has a 512-bit state which is initialized by copying into it a 128- or a 256-bit key, a 64-bit nonce and counter and a 128-bit constant. Previous attacks on Salsa used differential cryptanalysis exploiting a truncated differential over three or four rounds. The first attack was presented by Crowley [Cro06] which could break the 5 round version of Salsa20 within claimed 2^{165} trials. Later, a four round differential was exploited by Fischer et al. [FMB$^+$06] to break 6 rounds in 2^{177} trials and by Tsnunoo et al. [TSK$^+$07] to break 7 rounds in about 2^{190} trials. The currently best attack by Aumasson et al. [AFK$^+$08] covers an 8 round version of Salsa20 with an estimated complexity of 2^{251}.

The stream cipher Trivium was submitted by de Cannière and Preneel [dCP05] to the eSTREAM project [eST08] in 2005. Analog to Salsa20/12 Trivium is in the eSTREAM portfolio, even though it is in the hardware profile. Trivium uses an 80-bit key and an 80-bit initial value (IV) to generate up to 2^{64} bits of keystream. This synchronous cipher has a state size of 288 bits. The interesting part of Trivium is the non-linear update function of second degree. In [Rad06] Raddum presented and attacked simplified versions of Trivium called Bivium but the attack on Trivium had a complexity higher than the exhaustive key search. Bivium was completely broken by Maximov and Biryukov [MB07] and an attack on Trivium with complexity about 2^{100} was presented which showed that the key size of Trivium can not be increased just by loading longer keys into the state. In [MCP07], McDonald et al. attacked Bivium using SatSolvers. Another approach that gained attention recently is to reduce the key setup of Trivium as done by Turan and Kara [TK07] with 288 initialization rounds and Vielhaber [Vie07]

with 576 initialization rounds. Fischer, Khazaei and Meier [FKM08] showed an attack with complexity 2^{55} on a version with 672 initialization rounds. The best attack for Trivium known so far is the cube attack published by Dinur and Shamir [DS09] which can break a version with 767 initialization rounds with time complexity 2^{45}. So far, no attack faster than an exhaustive key search was shown for Trivium with full 1152 initialization rounds. A summary of attacks on Trivium is given in Table 4.1.

Attack authors	complexity
linearization technique to reduce the quadratic system factor c to solve the system of linear equations Maximov and Biryukov [MB07]	$c \cdot 2^{83.5}$
linear approximation with bias 2^{-31} 288 initialization rounds Turan and Kara [TK07]	
algebraic IV differential attack 576 initialization rounds, 47 recovered key bits Vielhaber [Vie07]	negligible
chosen IV statistical analysis key-recovery attack 672 initialization rounds Fischer, Khazei and Meier [FKM08]	2^{55}
cube attack 767 initialization rounds Dinur and Shamir [DS09]	2^{45}

Table 4.1: Attacks on Trivium

In this chapter we show a sliding property for the stream ciphers Salsa20 and Trivium. Slide attacks were introduced by Biryukov and Wagner [BW99, BW00] mainly to attack block ciphers but the method is applicable on stream ciphers as well.

We start with our investigation of Salsa20 followed by a description of the attacks. We show that the following observation holds: Suppose that two black boxes are given, one with Salsa20 and one with a random mapping. The attacker is allowed to choose a relation $\boldsymbol{\mathcal{F}}$ for a pair of inputs, after which a secret initial input x is chosen and a pair $(x, \boldsymbol{\mathcal{F}}(x))$ is encrypted either by Salsa20 or by a random mapping. We stress that only the relation $\boldsymbol{\mathcal{F}}$ is known to the attacker and no restrictions are made on the initial input. The goal of the attacker is given a pair of ciphertexts to tell whether they were encrypted by Salsa20 or by a random mapping. It is clear that for a truly random mapping no useful relation $\boldsymbol{\mathcal{F}}$ would exist. In contrast, Salsa20 can easily be distinguished from random if $\boldsymbol{\mathcal{F}}$ is a carefully selected function

related to the round-structure of Salsa20. If the initial input satisfies the Salsa20 conditions (secret key and known nonce, counter and constant), it is not only a distinguishing but also a complete key-recovery attack. To give the attacker a hard time the pair may be hidden in a large collection of other ciphertexts. For a random function there is no way of checking a large list except for checking all the pairs or doing a birthday attack. We describe for Salsa20 how to find such a hidden pair in a large list efficiently. Our attacks are independent of the number of rounds in Salsa and thus work for all three versions of Salsa. We also show a general birthday attack on 256-bit key Salsa20 with complexity 2^{192} which can be further sped up twice using sliding observations. Although as Bernstein [Ber08] mentioned there are better attacks on Salsa20 using parallelization, we wanted to point out that the sliding property is useful is this general birthday attack scenario.

In the second part of this chapter we describe our results on Trivium which show a large related key class (2^{39} out of 2^{80} keys) which produce identical keystreams up to a shift. We solve the resulting non-linear sliding equations using Magma and present several examples of such slid key-IV pairs. The interesting observation is that 24 key bits do not appear in the equations for a shift of 111 clocks and thus for a fixed IV, there is a 2^{24} freedom of choice for the key that the key-IV pair may have a sliding property.

4.1 Slid Pairs in Salsa20

In this section the symbol "$+$" denotes the addition modulo 2^{32}, the other two symbols work at the level of the bits with "$\oplus$" as XOR-addition and "$\lll$" as a shift of bits.

4.1.1 Brief Description of Salsa20

The Salsa20 encryption function uses the Salsa20 core function in a counter mode. The internal state of Salsa20 is a 4×4 - matrix of 32-bit words which is denoted by

$$Y = \begin{pmatrix} y_0 & y_1 & y_2 & y_3 \\ y_4 & y_5 & y_6 & y_7 \\ y_8 & y_9 & y_{10} & y_{11} \\ y_{12} & y_{13} & y_{14} & y_{15} \end{pmatrix} .$$

A so-called *quarterround* transforms an arbitrary vector (y_a, y_b, y_c, y_d) of four words into a vector (z_a, z_b, z_c, z_d) by calculating

$$\begin{aligned} z_b &= y_b \oplus ((y_a + y_d) \lll 7) \\ z_c &= y_c \oplus ((z_b + y_a) \lll 9) \\ z_d &= y_d \oplus ((z_c + z_b) \lll 13) \\ z_a &= y_a \oplus ((z_d + z_c) \lll 18) \ . \end{aligned}$$

This non-linear operation is the basic part of the *columnround* and the *rowround*. The columnround converts the matrix column by column using the quarterround by slightly shifting the entries of the columns as input for the quarterround. The shifted columns as input vectors for the quarterround are (y_0, y_4, y_8, y_{12}), (y_5, y_9, y_{13}, y_1), $(y_{10}, y_{14}, y_2, y_6)$ and $(y_{15}, y_3, y_7, y_{11})$. Similarly, the rowround transforms the matrix row by row using the quarterround by slightly shifting the entries of the rows as input for the quarterround. The shifted rows as input vectors for the quarterround are (y_0, y_1, y_2, y_3), (y_5, y_6, y_7, y_4), $(y_{10}, y_{11}, y_8, y_9)$ and $(y_{15}, y_{12}, y_{13}, y_{14})$. A so-called *doubleround* consists of a columnround followed by a rowround. The doubleround function of Salsa20 is repeated 10 times. If Y denotes the matrix, a keystream block is defined by

$$Z = Y + \text{doubleround}^{10}(Y) \ .$$

The quarterround has 12 word operations. One columnround as well as one rowround has 4 quarterrounds which means 48 word operations for each one. Salsa20 consists of 10 doublerounds which altogether gives 960 word operations. The feedforward at the end of Salsa20 has 16 word operations which concludes 976 word operations in total for one encryption.

The cipher takes as input a 256-bit key $(k_0, \ldots, k_7)$, a 64-bit nonce (n_0, n_1) and a 64-bit counter (c_0, c_1). The remaining four words have been set by the designer to fixed publicly known constants

$$\begin{aligned} \sigma_0 &= 0x61707865 & \sigma_1 &= 0x3320646e \\ \sigma_2 &= 0x79622d32 & \sigma_3 &= 0x6b206574 \ , \end{aligned}$$

given in hexadecimal numbers. Then a *starting state* S is given by the matrix

$$S = \begin{pmatrix} \sigma_0 & k_0 & k_1 & k_2 \\ k_3 & \sigma_1 & n_0 & n_1 \\ c_0 & c_1 & \sigma_2 & k_4 \\ k_5 & k_6 & k_7 & \sigma_3 \end{pmatrix} \ .$$

A 128-bit key version of Salsa20 copies the 128-bit key twice and has slightly different constants. In this chapter we mainly concentrate on the 256-bit key version.

4.1.2 Slid Pairs

The structure of a doubleround can be rewritten as columnround, then a matrix transposition, another columnround followed by a second transposition. We define $\boldsymbol{\mathcal{F}}$ to be a function which consists of a columnround followed by a transposition. Now the 10 doublerounds can be transferred into 20 times the function $\boldsymbol{\mathcal{F}}$. If we have two triples (k, n, c) and (k', n', c') with keys k, k', nonces n, n' and counters c, c' so that

$$\boldsymbol{\mathcal{F}}\Big[\ 1^{\text{st}} \text{ starting state } (k, n, c)\ \Big] \ = \ 2^{\text{nd}} \text{ starting state } (k', n', c')$$

then this property holds for each point during the round computation of Salsa20, especially for the end of the round computation. Please note that the feedforward at the end of Salsa20 destroys this property. We call such a pair of a 1st and 2nd starting state a *slid pair* and show their relation in Figure 4.1.

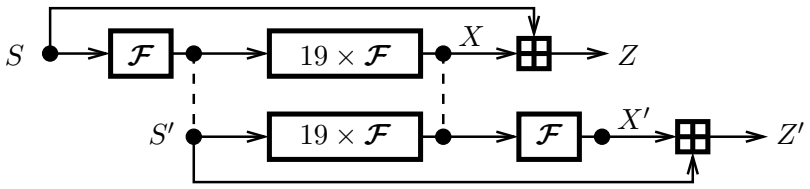

Figure 4.1: Relation of a slid pair for Salsa20

In a starting state four words are constants and 12 words can be chosen freely which leads to a total amount of 2^{384} possible starting states. If we want a starting state – after applying function $\boldsymbol{\mathcal{F}}$ – to result in a 2nd starting state, we obtain four wordwise equations. This means we can choose eight words of the 1st starting state freely whereas the other four words are determined by the equations as well as the words for the 2nd starting state. This leads to a total amount of 2^{256} possible slid pairs.

For the 128-bit key version no such slid pair exists due to the additional constraints of four fewer words freedom in the 1st starting state and four more wordwise equations in the 2nd starting state.

With function $\boldsymbol{\mathcal{F}}$ we get two equations $S' = \boldsymbol{\mathcal{F}}(S)$ and $X' = \boldsymbol{\mathcal{F}}(X)$. The words for these matrices we denote as

$$S = \begin{pmatrix} \sigma_0 & k_0 & k_1 & k_2 \\ k_3 & \sigma_1 & n_0 & n_1 \\ c_0 & c_1 & \sigma_2 & k_4 \\ k_5 & k_6 & k_7 & \sigma_3 \end{pmatrix} \qquad S' = \begin{pmatrix} \sigma_0 & k'_0 & k'_1 & k'_2 \\ k'_3 & \sigma_1 & n'_0 & n'_1 \\ c'_0 & c'_1 & \sigma_2 & k'_4 \\ k'_5 & k'_6 & k'_7 & \sigma_3 \end{pmatrix}$$

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} \qquad X' = \begin{pmatrix} x'_0 & x'_1 & x'_2 & x'_3 \\ x'_4 & x'_5 & x'_6 & x'_7 \\ x'_8 & x'_9 & x'_{10} & x'_{11} \\ x'_{12} & x'_{13} & x'_{14} & x'_{15} \end{pmatrix}$$

$$Z = \begin{pmatrix} z_0 & z_1 & z_2 & z_3 \\ z_4 & z_5 & z_6 & z_7 \\ z_8 & z_9 & z_{10} & z_{11} \\ z_{12} & z_{13} & z_{14} & z_{15} \end{pmatrix} \qquad Z' = \begin{pmatrix} z'_0 & z'_1 & z'_2 & z'_3 \\ z'_4 & z'_5 & z'_6 & z'_7 \\ z'_8 & z'_9 & z'_{10} & z'_{11} \\ z'_{12} & z'_{13} & z'_{14} & z'_{15} \end{pmatrix} .$$

The set up of the system of equations for a whole Salsa20 computation is too complicated but the equations for the computation of $\boldsymbol{\mathcal{F}}$ are very clear. For a complete description of the equations see Appendix A. The

structure of both systems of equations coming from the relation $\boldsymbol{\mathcal{F}}$ is the same, especially all the known variables are at the same place. Due to the eight words freedom we have in a 1^{st} or 2^{nd} starting state, there are some relations in the 12 non-fixed words. For the 2^{nd} starting state these relations are very clear as they only deal with words

$$0 = k'_2 + k'_1 \quad 0 = k'_3 + n'_1 \quad 0 = c'_1 + c'_0 \quad \text{and} \quad 0 = k'_7 + k'_6 \ . \tag{4.1}$$

If we have an arbitrary starting state and the equations (4.1) are satisfied we know this starting state is a 2^{nd} starting state. In contrast, for the 1^{st} starting state the conditions to be a 1^{st} starting state depend on the single bits and not on words anymore. Thus, the conditions for a 1^{st} starting state are more complicated and not obvious, hence we omit them. Sliding by the function $\boldsymbol{\mathcal{F}}$ is applicable to any version of Salsa20/r where r is even. For odd r there would be no transposition at the end of the round computation. Consequentially the equations are a bit different, though still solvable.

4.1.3 Sliding State Recovery Attack on the Davies-Meyer Mode

In this section we consider a general state-recovery slide attack on a Davies-Meyer construction. We demonstrate it with an example of a Davies-Meyer feedforward used with the iterative permutation of Salsa20. The feedforward breaks the sliding property and makes slide attack more complicated to mount. We consider the following scenario:

1. The oracle chooses a secret 512-bit state S (here we assume that there is no restriction of 128-bit diagonal constants and the full 512 bits can be chosen at random).
2. The oracle computes $\boldsymbol{\mathcal{F}}(S) = S'$.
3. The oracle computes $Z = \text{Salsa20}(S)$, $Z' = \text{Salsa20}(S')$ and gives them to the attacker.
4. The goal of the attacker is to recover the secret state S.

Due to the weak diffusion of $\boldsymbol{\mathcal{F}}$ we can write separate systems of equations for each column of S. If we combine for one column the quarterround coming from $S' = \boldsymbol{\mathcal{F}}(S)$, the corresponding quarterround from $X' = \boldsymbol{\mathcal{F}}(X)$ and the feedforward, we get a system with 16 equations shown below. We assume all 16 variables are unknown.

$$\begin{aligned} s'_1 &= s_4 \oplus ((s_0 + s_{12}) \lll 7) & \qquad x'_1 &= x_4 \oplus ((x_0 + x_{12}) \lll 7) \\ s'_2 &= s_8 \oplus ((s'_1 + s_0) \lll 9) & \qquad x'_2 &= x_8 \oplus ((x'_1 + x_0) \lll 9) \\ s'_3 &= s_{12} \oplus ((s'_2 + s'_1) \lll 13) & \qquad x'_3 &= x_{12} \oplus ((x'_2 + x'_1) \lll 13) \\ s'_0 &= s_0 \oplus ((s'_3 + s'_2) \lll 18) & \qquad x'_0 &= x_0 \oplus ((x'_3 + x'_2) \lll 18) \end{aligned}$$

$$\begin{array}{lll} z_0 &= x_0 + s_0 & z'_0 = x'_0 + s'_0 \\ z_4 &= x_4 + s_4 & z'_1 = x'_1 + s'_1 \\ z_8 &= x_8 + s_8 & z'_2 = x'_2 + s'_2 \\ z_{12} &= x_{12} + s_{12} & z'_3 = x'_3 + s'_3 \end{array}$$

This system can be reduced to four equations.

$$\begin{aligned} z'_1 &= \Big[(z_4 - s_4) \oplus \Big([(z_0 - s_0) + (z_{12} - s_{12})] \lll 7\Big)\Big] \\ &\quad + \Big[s_4 \oplus \Big((s_0 + s_{12}) \lll 7\Big)\Big] \end{aligned} \tag{4.2}$$

$$\begin{aligned} z'_2 &= \Big[(z_8 - s_8) \oplus \Big([(z'_1 - s'_1) + (z_0 - s_0)] \lll 9\Big)\Big] \\ &\quad + \Big[s_8 \oplus \Big((s'_1 + s_0) \lll 9\Big)\Big] \end{aligned} \tag{4.3}$$

$$\begin{aligned} z'_3 &= \Big[(z_{12} - s_{12}) \oplus \Big([(z'_2 - s'_2) + (z'_1 - s'_1)] \lll 13\Big)\Big] \\ &\quad + \Big[s_{12} \oplus \Big((s'_2 + s'_1) \lll 13\Big)\Big] \end{aligned} \tag{4.4}$$

$$\begin{aligned} z'_0 &= \Big[(z_0 - s_0) \oplus \Big([(z'_3 - s'_3) + (z'_2 - s'_2)] \lll 18\Big)\Big] \\ &\quad + \Big[s_0 \oplus \Big((s'_3 + s'_2) \lll 18\Big)\Big] \end{aligned} \tag{4.5}$$

In (4.2) we guess two variables out of s_4, s_0, s_{12} and calculate the third one. With these three variables we calculate s'_1 and solve (4.3) to get s_8. Now we can compute s'_2, s'_3, s'_0 and check our guess with (4.4) and (4.5). Therefore, the system of equations for one column with completely unknown variables can be solved by guessing only two variables. Similarly, we solve the systems of equations for the other three columns. With the four guesses of 2^{64} steps each we can completely recover the 512-bit secret state S. This shows that Salsa20 without the diagonal constants is easily distinguishable from a random function, for which a similar task would require about 2^{511} steps.

Now we add the diagonal constants which reduces the flexibility of the oracle in a choice of the initial states to 2^{256} but the attack works even better:

1. The oracle chooses a starting state S' with key k', nonce n' and counter c' satisfying equations (4.1). The attacker does not know this state.

2. The oracle applies $\boldsymbol{\mathcal{F}}^{-1}(S')$ to compute the related key k, nonce n and counter c for the starting state S.

3. The oracle computes $Z = \text{Salsa20}(S)$, $Z' = \text{Salsa20}(S')$ and gives them to the attacker.

4. The goal of the attacker is to recover the secret state S.

The knowledge of the diagonal makes the previous attack even faster and allows the full 384-bit (256-bit entropy) state recovery with complexity of $4 \cdot 2^{32}$ because in the system of equations for each column only one variable has to be guessed. If the attacker chooses the nonce and the counter n', c' (160-bit entropy) then the complexity drops to $2 \cdot 2^{32}$. Furthermore, if nonce and counter n, c are known (128-bit entropy left), the state can be recovered immediately (with or without knowing nonce and counter n', c') as shown in the following section. Table 4.2 shows the time complexities for the described attacks, memory complexity is negligible.

known words of the starting state	sliding on Salsa20	random oracle
nothing	2^{66}	2^{511}
only diagonal	2^{34}	2^{255}
diagonal, n', c'	2^{33}	2^{159}
diagonal, n, c	directly	2^{127}
diagonal, n, c and n', c'	directly	2^{63}

Table 4.2: Time complexities for state-recovery attacks

4.1.4 Related Key Key-Recovery Attack on Salsa20

Let us assume that we know the two ciphertexts and the corresponding nonces and counters of a slid pair. We do not know both keys but we know that both starting states differ by function $\boldsymbol{\mathcal{F}}$ which gives the relation shown in Figure 4.2 and the fact that the 2nd starting state conforms to the initial state format of Salsa20 (i.e. has proper 128-bit constant on the diagonal). In this section we show that this information is sufficient to completely recover the two related 256-bit secret keys of Salsa20 immediately.

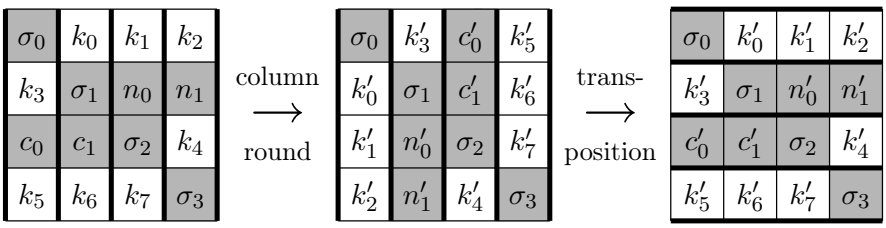

Figure 4.2: Relation of the 1st and 2nd starting state (The grey squares are the known words.)

With the knowledge of both nonces and counters we are able to recover four words of the first unknown key and two words of the second unknown

key (marked by bold letters in the following) by solving the equations[1]

$$\begin{aligned} \boldsymbol{k'_3} &= -n'_1 & \boldsymbol{k_0} &= k'_3 \oplus ((n'_1 + n'_0) \lll 13) \\ \boldsymbol{k_6} &= n'_1 \oplus ((n'_0 + \sigma_1) \lll 9) & \boldsymbol{k'_4} &= -c'_0 + ((c'_1 \oplus n_0) \ggg 13) \\ \boldsymbol{k_1} &= c'_0 \oplus ((k'_4 + \sigma_2) \lll 9) & \boldsymbol{k_7} &= k'_4 \oplus ((\sigma_2 + n_0) \lll 7) \ . \end{aligned}$$

At this point we still have not used the similar equations from the bottom of the round computation. If parts of the nonce / counter are unknown these equations can be used to check our guesses.

Then two similar systems of equations are left to get the other 10 key words. For the first one we solve the equation

$$z'_2 = [(z_8 - c_0) \oplus ((z'_1 - \boldsymbol{k'_0} + z_0 - \sigma_0) \lll 9)] + [c_0 \oplus ((\boldsymbol{k'_0} + \sigma_0) \lll 9)] \ , \quad (4.6)$$

whereas some words of the ciphertexts are used and only k'_0 is unknown. With the solution of k'_0 we get four more words

$$\begin{aligned} \boldsymbol{k'_1} &= c_0 \oplus ((k'_0 + \sigma_0) \lll 9) & \boldsymbol{k'_2} &= -k'_1 \\ \boldsymbol{k_5} &= k'_2 \oplus ((k'_1 + k'_0) \lll 13) & \boldsymbol{k_3} &= k'_0 \oplus ((\sigma_0 + k_5) \lll 7) \ . \end{aligned}$$

For the second system we solve a similar equation to get k'_5

$$z'_{13} = [(z_7 - n_1) \oplus ((z'_{12} - \boldsymbol{k'_5} + z_{15} - \sigma_3) \lll 9)] + [n_1 \oplus ((\boldsymbol{k'_5} + \sigma_3) \lll 9)] \ . \quad (4.7)$$

With the solution of k'_5 we get the last four words

$$\begin{aligned} \boldsymbol{k'_6} &= n_1 \oplus ((k'_5 + \sigma_3) \lll 9) & \boldsymbol{k'_7} &= -k'_6 \\ \boldsymbol{k_4} &= k'_7 \oplus ((k'_6 + k'_5) \lll 13) & \boldsymbol{k_2} &= k'_5 \oplus ((\sigma_3 + k_4) \lll 7) \ . \end{aligned}$$

Equation (4.6) (similarly (4.7)) can be solved by just checking all 2^{32} possibilities for k'_0 (respectively k'_5 for (4.7)), however it can be done much more efficiently with less then 2^8 steps by guessing k'_0 (or k'_5) gradually and checking the bitwise equations. We solved 16 equations but used 26 of the 64 equations in Appendix A to get these 16 equations. The complexity to get all the 16 key words is approximately 56 word operations which is much faster than a single Salsa20 encryption (976 word operations). We compute Salsa20 for both keys and compare the calculated ciphertexts to the given ones. If they match we have found the right keys.

4.1.5 A Generalized Related Key Attack on Salsa20

Suppose we are given a (possibly large) list of ciphertexts with the corresponding nonces and counters and we are told that the slid pair is hidden in this list. The question is: Can we efficiently find slid pairs in a large list of

[1] The complete system of equations is shown in Appendix A, here we only give the rearranged equations.

ciphertexts? As we saw in the previous section, it is easy to compute both keys given such slid ciphertext pairs. The task is made more difficult by the feedforward of Salsa20 which destroys the sliding relationship. Nevertheless, in this section we show that given a list of ciphertexts of size $O(2^l)$ it is possible to detect a slid pair with memory and time complexity of just $O(2^l)$.[2] The naive approach, which would require to check the equations from function $\boldsymbol{\mathcal{F}}$ for each possible pair, will have complexity $O(2^{2l})$ which is too expensive. Our idea is to reduce the amount of potential pairs by sorting them by eight precomputed words, so that only elements where these eight words match have the possibility to yield a slid pair. After decreasing the number of possible pairs in that way we can check the remaining pairs using additional constraints coming from the sliding equations.

For the sorting we use bucket sort because each word has only 2^{32} possibilities. The number of words we sort by is equal to the number of runs of bucket sort.

We have a set M of ciphertexts with corresponding nonces and counters. Each ciphertext can be either a 1^{st} starting state or a 2^{nd} starting state. To regard this, the set is stored twice, first under M_1 to check for possible 1^{st} starting states and second under M_2 to check for possible 2^{nd} starting states.

Step 1: **Sort the first list**
For each element in set M_1, undo the feedforward for the four words on the diagonal and for $x_9 = z_9 - c_1$. Then sort M_1 by the specified eight words $x_0, x_5, x_{10}, x_{15}, x_9$ and c_1, z_1, z_{13}.

Step 2: **Sort the second list**[3]
Select only elements of M_2 that satisfy equation $0 = c'_0 + c'_1$ since only such an entry can be a 2^{nd} starting state. For each element undo the feedforward for the four words on the diagonal and for $x'_6, \ldots, x'_9$ because nonces and counters are known. Then for each element compute the words marked in bold in the equations

$$
\begin{aligned}
\boldsymbol{x_0} &= x'_0 \oplus ((z'_2 + z'_3) \lll 18) & \qquad \boldsymbol{k'_3} &= -n'_1 \\
\boldsymbol{x_5} &= x'_5 \oplus ((z'_4 + n'_1 + x'_7) \lll 18) & \qquad \boldsymbol{x_{10}} &= x'_{10} \oplus ((x'_9 + x'_8) \lll 18) \\
\boldsymbol{x_{15}} &= x'_{15} \oplus ((z'_{13} + z'_{14}) \lll 18) & \qquad \boldsymbol{x_1} &= (z'_4 + n'_1) \oplus ((x'_7 + x'_6) \lll 13) \\
\boldsymbol{x_9} &= x'_6 \oplus ((x_5 + x_1) \lll 7) & \qquad \boldsymbol{k_0} &= k'_3 \oplus ((n'_1 + n'_0) \lll 13) \\
\boldsymbol{c_1} &= n'_0 \oplus ((\sigma_1 + k_0) \lll 7) & \qquad \boldsymbol{z_1} &= x_1 + k_0 \\
\boldsymbol{k_6} &= n'_1 \oplus ((n'_0 + \sigma_1) \lll 9) & \qquad \boldsymbol{z_{13}} &= (k_6 + x'_7) \oplus ((x'_6 + x_5) \lll 9) \ .
\end{aligned}
$$

[2] Sorting is done via bucket sort so we save the logarithmic factor l in complexity.

[3] If the number of rounds of Salsa is odd then such simple sorting would not be possible since Salsa equations are easier to solve in reverse direction. In our approach we know two words at the input and three words at the output of the columnround which is easier to solve than the opposite (three words at the input vs. two at the output). Nevertheless, the system is still solvable.

During this computation we calculate three key words k_0, k_6 and k'_3. Sort the set M_2 by the calculated eight words $x_0, x_5, x_{10}, x_{15}, x_9$ and c_1, z_1, z_{13} for the potential 1st starting states.

Step 3: **Check each possible pair**
Cross check all the possible pairs that match in these eight words and thus satisfy the 256-bit filtering. For the conforming pairs we can continue the check using the equations below. If a test condition is wrong this pair can not be a slid pair. For each pair undo the feedforward for the word $x_6 = z_6 - n_0$ of the ciphertext of the 1st starting state. Then compute the bold variables and check the conditions for the three equations

$$\begin{aligned}
\text{compute} \quad & \boldsymbol{k'_4} = -c'_0 + ((n_0 \oplus c'_1) \ggg 13) \\
& \boldsymbol{x'_{11}} = -x'_8 + ((x_6 \oplus x'_9) \ggg 13) \\
\text{check} \quad & z'_{11} = x'_{11} + k'_4 \\
\text{compute} \quad & \boldsymbol{k_1} = c'_0 \oplus ((k'_4 + \sigma_2) \lll 9) \\
& \boldsymbol{x_2} = x'_8 \oplus ((x'_{11} + x_{10}) \lll 9) \\
\text{check} \quad & z_2 = x_2 + k_1 \\
\text{compute} \quad & \boldsymbol{k_7} = k'_4 \oplus ((\sigma_2 + n_0) \lll 7) \\
& \boldsymbol{x_{14}} = x'_{11} \oplus ((x_{10} + x_6) \lll 7) \\
\text{check} \quad & z_{14} = x_{14} + k_7 \ .
\end{aligned}$$

During this computation we calculate the three key words k_1, k_7 and k'_4.

For the remaining pairs we have two further systems of equations to check. For the first one we solve (4.6) and if there is no solution for k'_0 this pair can not be a slid pair. Otherwise, we use k'_0 to compute four additional key words while we check two more conditions

$$\begin{aligned}
\text{compute} \quad & \boldsymbol{k'_1} = c_0 \oplus ((k'_0 + \sigma_0) \lll 9) \\
& \boldsymbol{k'_2} = -k'_1 \\
& \boldsymbol{k_5} = k'_2 \oplus ((k'_1 + k'_0) \lll 13) \\
& \boldsymbol{x'_1} = z'_1 - k'_0 \\
& \boldsymbol{x_{12}} = (z'_3 - k'_2) \oplus ((z'_2 - k'_1 + x'_1) \lll 13) \\
\text{check} \quad & z_{12} = x_{12} + k_5 \\
\text{compute} \quad & \boldsymbol{k_3} = k'_0 \oplus ((\sigma_0 + k_5) \lll 7) \\
& \boldsymbol{x_4} = x'_1 \oplus ((x_0 + x_{12}) \lll 7) \\
\text{check} \quad & z_4 = x_4 + k_3 \ .
\end{aligned}$$

For the second system we similarly solve (4.7) and again if there is no solution for k'_5 this pair can not be a slid pair. Otherwise, we use k'_5 to

compute the rest of the key words while we check two more conditions

$$
\begin{aligned}
\text{compute} \quad & \boldsymbol{k'_6} = n_1 \oplus ((k'_5 + \sigma_3) \lll 9) \\
& \boldsymbol{k'_7} = -k'_6 \\
& \boldsymbol{k_4} = k'_7 \oplus ((k'_6 + k'_5) \lll 13) \\
& \boldsymbol{x'_{12}} = z'_{12} - k'_5 \\
& \boldsymbol{x_{11}} = (z'_{14} - k'_7) \oplus ((z'_{13} - k'_6 + x'_{12}) \lll 13) \\
\text{check} \quad & z_{11} = x_{11} + k_4 \\
\text{compute} \quad & \boldsymbol{k_2} = k'_5 \oplus ((\sigma_3 + k_4) \lll 7) \\
& \boldsymbol{x_3} = x'_{12} \oplus ((x_{15} + x_{11}) \lll 7) \\
\text{check} \quad & z_3 = x_3 + k_2 \ .
\end{aligned}
$$

We have nine extra test conditions to check the potential slid pairs. We would only expect seven conditions but due to the different arithmetic operations the dependencies of the equations are not clear. In total, we have at least filtering power of $32 \cdot 7$ bits. Thus, we expect that only the correct slid pairs survive this check. The remaining pairs are the correct slid pairs for which we completely know both keys.

Complexity.

Assuming we are given a list of 2^l ciphertexts with corresponding nonces and counters. Instead of storing the list twice we use two kinds of pointers, one kind for the potential 1^{st} starting states and the other one for the potential 2^{nd} starting states. For the pointers we need $\frac{l}{32} \cdot 2^l$ words of memory. A summary for the complexity of different lists is given in Table 4.3. The larger the list of the random states in which our target is hidden – the larger would be the complexity of the attack. However, the time complexity of the attack grows only linearly with the size of the list.

list size	memory in words	time in Salsa20 clocks
2^{128}	$28 \cdot 2^{128}$	2^{122}
2^{192}	$32 \cdot 2^{192}$	2^{186}
2^{256}	$36 \cdot 2^{256}$	2^{250}

Table 4.3: Complexities for different list sizes

The number of slid pairs is 2^{256} which gives the probability of 2^{-128} for a random starting state to be a 1^{st} or a 2^{nd} starting state. Via the birthday paradox we expect the amount of 2^{256} random ciphertexts to contain both stating states for one slid pair. We have described how to search in a big list efficiently for a slid pair and recover both secret keys.

4.1.6 Time-Memory Tradeoff Attacks on Salsa

Salsa20 has 2^{384} possible starting states. We notice that the square root of 2^{384} is less than the keyspace size for keys longer than 192 bits. Thus, a trivial birthday attack on 256-bit key Salsa20 would proceed as follows:

During the preprocessing stage, we generate a list of 2^{192} randomly chosen starting states and run Salsa20 for each of them to get a sample of ciphertexts. Afterwards, we sort this list by the ciphertexts. During the online stage, we capture 2^{192} ciphertexts for which we want to find the keys. We do not have to store these ciphertexts and can check each of them immediately for a match with the sorted array of precomputed ciphertexts. If we have a match we retrieve the corresponding key from our table. Of course, due to very high memory complexity this attack can only be viewed as a certificational weakness.

If we can choose the nonce or the counter there are only 2^{320} different starting states reducing the attack to precomputation and memory complexity of 2^{160} and – if we can choose both – the state space drops to 2^{256} and the attack complexity drops to 2^{128}. Similar reasoning for 128-bit key Salsa20 would yield an attack with 2^{64} complexity. Thus, it is crucial for the security of Salsa20 that nonces are chosen at random for each new key and the counter is not stuck at some fixed value (like 0, for example).

The complexities are summarized in Table 4.4 where R stands for a complete run of Salsa20 and M for a matrix of Salsa (16 words).

attack	precomputation	memory	time	captured ciphertext
chosen nonce and counter	$R \cdot 2^{128}$	$2M \cdot 2^{128}$	2^{128}	2^{128}
chosen nonce or counter	$R \cdot 2^{160}$	$2M \cdot 2^{160}$	2^{160}	2^{160}
general	$R \cdot 2^{192}$	$2M \cdot 2^{192}$	2^{192}	2^{192}
using sliding property	$R \cdot 2^{192}$	$2.5M \cdot 2^{192}$	2^{192}	2^{191}

Table 4.4: Complexities for the Birthday attack

Improved Birthday Using the Sliding Property.

We can use the sliding property to increase the efficiency of the birthday attack twice (which can be translated into the reduction of memory or time, or the increase of success probability of the birthday attack).

Salsa20 has 2^{384} possible starting states in total and the sliding property reduces the number of possible starting states to 2^{257} (a slid pair has two starting states). Hence, a random starting state has the probability of 2^{-127} to be a starting state for a slid pair (either 1^{st} or 2^{nd} one).

During the preprocessing stage we generate a sample of 2^{192} 2^{nd} starting states by using (4.1) of page 34 and choose the remaining eight words at random. We compute the corresponding ciphertexts for these states as well as the eight specified words for the corresponding 1^{st} starting states as mentioned in step 2 of Section 4.1.5. We use two kinds of pointers to sort this generated list by the ciphertexts for the 2^{nd} starting states and by the eight words for the corresponding 1^{st} starting state. We capture ciphertext from the keystream where we also know the nonce and the counter. The amount of 2^{191} captured ciphertexts contains about either 2^{64} 1^{st} starting states or 2^{64} 2^{nd} starting states. We check if the ciphertext is a correct one for a 2^{nd} starting state from our list (direct birthday) or matches the eight words for a 1^{st} starting state for one of the states from our collection (then proceed with step 3 as described in Section 4.1.5 to check the remaining eight words) (indirect birthday). In both cases we obtain the key for this ciphertext.

4.2 Slid Pairs in Trivium

4.2.1 Brief Description of Trivium

The designers introduced the stream cipher Trivium with a state size of 288 bits. This internal state can be split into three registers as shown in Figure 4.3. The first register which we call A has a length of 93, the second one called B has a length of 84 and the last register named C has 111 bits. The internal state is denoted in the following way:

$$\text{A: } (s_1, s_2, \ldots, s_{93}) \quad \text{B: } (s_{94}, s_{95}, \ldots, s_{177}) \quad \text{C: } (s_{178}, s_{279}, \ldots, s_{288}) \ .$$

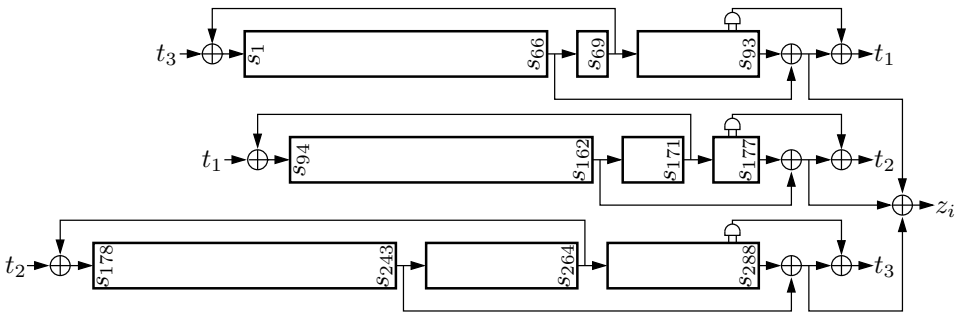

Figure 4.3: Trivium with three registers

Update and Keystream Production.

The non-linear update function of second degree uses 15 bits of the internal state to compute three new bits, each for one register, and the keystream bit z_i is calculated by summing up only 6 of these 15 bits. In the following pseudo-code all computations are over GF(2).

$$
\begin{aligned}
t_1 &\longleftarrow s_{66} + s_{93} \\
t_2 &\longleftarrow s_{162} + s_{177} \\
t_3 &\longleftarrow s_{243} + s_{288} \\
z_i &\longleftarrow t_1 + t_2 + t_3 \\
t_1 &\longleftarrow t_1 + s_{91} \cdot s_{92} + s_{171} \\
t_2 &\longleftarrow t_2 + s_{175} \cdot s_{176} + s_{264} \\
t_3 &\longleftarrow t_3 + s_{286} \cdot s_{287} + s_{69}
\end{aligned}
$$

$$
\begin{aligned}
&\text{A:} \quad (s_1, s_2, ..., s_{93}) &&\longleftarrow (t_3, s_1, \ldots, s_{92}) \\
&\text{B:} \quad (s_{94}, s_{95}, \ldots, s_{177}) &&\longleftarrow (t_1, s_{94}, \ldots, s_{176}) \\
&\text{C:} \quad (s_{178}, s_{279}, \ldots, s_{288}) &&\longleftarrow (t_2, s_{178}, \ldots, s_{287})
\end{aligned}
$$

Key and IV setup.

In register A the key of 80 bits is loaded and the last 13 bits are set to zero. The IV is loaded into the first 80 bits of register B and the last 4 bits are set to zero. In register C all positions are set to zero except for the last three bits which are set to one.

$$
\begin{aligned}
&\text{A:} \quad (s_1, s_2, ..., s_{93}) &&\longleftarrow (K_{80}, \ldots, K_1, 0, \ldots, 0) \\
&\text{B:} \quad (s_{94}, s_{95}, \ldots, s_{177}) &&\longleftarrow (IV_{80}, \ldots, IV_1, 0, \ldots, 0) \\
&\text{C:} \quad (s_{178}, s_{279}, \ldots, s_{288}) &&\longleftarrow (0, \ldots, 0, 1, 1, 1)
\end{aligned}
$$

In this chapter we refer to this state with key, IV and 128 fixed positions as starting state. After the registers are initialized in the described way, the cipher is clocked $4 \cdot 288$ times using the update function without producing any keystream bits. This will finish the key setup. Now each following clock will produce a keystream bit.

4.2.2 Slid Pairs

We start with the observation made by Jin Hong on the eSTREAM forum [Hon05] that it is possible to produce sliding states in Trivium. We searched for pairs of key and IV which produce another starting state after a few clocks. If we have a key and IV pair (K_1, IV_1) which produces another starting state with a key and IV (K_2, IV_2), the keystream created by (K_2, IV_2) will be the same as the one created by (K_1, IV_1) except for a shift of some bits. The number of shifted bits is equal to the number of clocks needed to get from the 1^{st} to the 2^{nd} starting state. We call such a pair of

two key and IV pairs a *slid pair* and denote this with $[(K_1, IV_1), (K_2, IV_2), c]$ whereas c stands for the number of clock-shifts.

Due to the special structure of the third register with 108 zeros and the last three ones, the first possibility of a 2^{nd} starting state to occur is after 111 clocks. Each following clock gives the chance for a 2^{nd} starting state. Table 4.5 shows two examples for slid pairs with different shifts written in hexadecimal numbers. The bits for keys and IVs are ordered from 1 to 80 but in the key-IV setup they are used the other way around.

$[(K_1, IV_1), (K_2, IV_2), 111]$	
K_1 :	70011000001E00000000
IV_1 :	AF9D635BCEF9AE376CF7
keystream[4]:	2E7338CB404272ABEE3F7BEC2F8D
	55E27536D29AFFFF15DFDFD711AECC78D13D7B61 ...
K_2 :	780000001DA2000003C1
IV_2 :	1DF35CF6D4FFF4E3A6C0
keystream:	55E27536D29AFFFF15DFDFD711AECC78D13D7B61 ...

$[(K_3, IV_3), (K_4, IV_4), 112]$	
K_3 :	02065B9C001730000000
IV_3:	609FC141828705160A3C
keystream:	A48BCA9143685F03DE646F83AB52
	88BC9542798983349A959503E63BBF29C4755DE6 ...
K_4 :	B98000003E96E70005CE
IV_4 :	2B7C1483BC476A62E4CB
keystream:	88BC9542798983349A959503E63BBF29C4755DE6 ...

Table 4.5: Two examples for slid pairs with different shifts

4.2.3 Systems of Equations

We describe the 2^{nd} starting state as polynomial equations in the 80 key and 80 IV variables of the 1^{st} pair. The 128 fixed positions in a starting state yield a system of equations with 160 variables and 128 equations. We have more variables than equations which results in 2^{32} solutions. To solve these systems we tried the F4 algorithm implemented in the computer algebra system Magma to get a Gröbner basis and the solutions for (K_1, IV_1) but gave up after $c = 115$ because of the long computation time. A more brute force approach – guessing a part of the variables, checking this guess and outputting the solution – which we implemented for each individual c worked

[4]The shift is $c = 111$ which means the first 111 bits are a prefix. When rewriting this prefix from hexadecimal to binary numbers the leading zero must be omitted because 111 is not a multiple of 4.

much better. To get the 2^{nd} key and IV we can use the systems of equations which describe the key and IV in the 2^{nd} starting state just inserting the known values of the 1^{st} key and IV pair or simply clock Trivium c times starting from (K_1, IV_1) to get (K_2, IV_2).

Some Facts about these Systems.

The system of equations for the first instance which appears after 111 clocks contains only 136 variables because the last 24 bits of the key do not occur in this system. Furthermore, 16 bits are given a priori due to the 13 zeros in register A and 3 ones in register C. The degree of the monomials in the equations is raised from 1 to 3. Due to the 24 key bits missing in the equations, these bits can be chosen arbitrarily. This leads us to 2^{24} different keys for one IV in the 1^{st} key and IV pair of a slid pair. Table 4.6 collects some facts for the systems which we solved with our brute force approach.

clock-shift c	111	112	113	114	115	116	$\cdots$	124
variables in equations	136	137	138	139	140	141	$\cdots$	149
last key bits not in the equation	24	23	22	21	20	19	$\cdots$	11
a priori given bits	16	15	14	13	13	13	$\cdots$	13
computing time for Magma in days	2.5	2.5	10	32.5	64	–	–	–
guess bits for Magma[5]	0	4	6	8	10	–	–	–

Table 4.6: Some facts for the systems of equations

We found that we sometimes have slightly less but mostly have slightly more solutions that we would have expected as shown in Table 4.7.

The higher the clock-shift will be, the more complicated the systems of equations will become. For each clock-shift another system of equations is needed but for every step of c most of the equations are the same or related. Due to the length of register C which defines the occurrence of a second starting state we have at least 111 clock-shifts. Thus, we expect a minimum $111 \cdot 2^{32} \approx 2^{39}$ slid pairs just within a shift of 221 bits of each other. There are much more slid pairs for longer shifts but the equations would be much more complicated.

Non-existence of Special Slid Pairs.

We searched for slid pairs with additional constraints. The first type applies when the keys in both key and IV pairs are the same for any clock-shift c: $([(K, IV_1), (K, IV_2)], c)$, and the second type applies when both times the

[5]We guessed these bits to get a solution from Magma in a reasonable amount of time.

c	expected	real solutions
111	2^8	$2^8 + 84$
112	2^9	$2^9 - 72$
113	2^{10}	$2^{10} - 176$
114	2^{11}	$2^{11} + 2^9$
115	2^{12}	$2^{12} - 2^8$
116	2^{13}	$2^{13} - 2^7$
117	2^{14}	$2^{14} - 2^9$
118	2^{15}	$2^{15} + 2^{12} + 2^{10} + 2^9$
119	2^{16}	$2^{16} + 2^{13}$
120	2^{17}	$2^{17} - 2^{12}$
121	2^{18}	$2^{18} + 2^{14} + 2^{12} + 2^{11}$
122	2^{19}	$2^{19} + 2^{14} + 2^{13} + 2^{12} + 2^{10}$
123	2^{20}	$2^{20} + 2^{15}$
124	2^{21}	$2^{21} + 2^{13} + 2^{10} + 2^9$

Table 4.7: Some facts for solutions of different systems of equations

same IV is used for any clock-shift c: $([(K_1, IV), (K_2, IV)], c)$. In both cases the fixed 2^{nd} key or IV leads to 80 additional equations which account for the occurrence of all 80-bit of key or IV resulting in overdefined systems with 208 equations and 160 variables. For both types the systems are not likely to be solvable for any reasonably small amount of shift. As a result of the 48 extra equations the chance for such a system to have a solution is about 2^{-48}. We computed that for the first 31 instances (clock-shifts 111 up to 142) these systems have no solution.

4.3 Summary

In this chapter we have described sliding properties of Salsa20 and Trivium which lead to distinguishing, key-recovery and related key attacks on these ciphers. We have also shown that Salsa20 does not offer 256-bit security due to a simple birthday attack on its 384-bit state. Since the likelihood of falling in our related key classes by chance is relatively low (2^{256} out of 2^{384} for Salsa20, 2^{39} out of 2^{80} for Trivium), these attacks do not threaten most of the real-life usage of these ciphers. However, designers of protocols which would use these primitives should definitely be aware of these non-randomness properties which can be exploited in certain scenarios.

In 2011 Bernstein [Ber11] published a modified version of Salsa20 called XSalsa20 which uses a 192-bit nonce. XSalsa20 uses the Salsa20 matrix and the Salsa20 computations but performs one step prior to the keystream generation. For this step the matrix is loaded with the constants, the key and the first 128 bits of the 192-bit nonce which are placed at the positions of the

former 64-bit nonce and 64-bit counter. Thereafter, Salsa20 is performed on this matrix without the feedforward at the end. Eight words are taken from the output matrix, four words at the diagonal and four words at the positions of the former 64-bit nonce and 64-bit counter. This concludes the prior step and these eight words are used as key for the keystream generation. Now XSalsa20 behaves like Salsa20 and the matrix is loaded with the constants, the eight specified words as key, the last 64 bits of the 192-bit nonce as 64-bit nonce and the 64-bit counter. The sliding property of Salsa20 is preserved for the keystream generation of XSalsa20 but not for the prior step. Thus, we can compute the starting matrix for the keystream generation but not the key itself.

Chapter 5

Analysis of SNOW 3G$^{\oplus}$ and SNOW 2.0$^{\oplus}$ Resynchronization Mechanism

The stream cipher SNOW 3G is the core of the 3GPP confidentiality and integrity algorithms UEA2 and UIA2, published in 2006 by the 3GPP Task Force [ETS06a]. Compared to its predecessor SNOW 2.0 [EJ02], SNOW 3G adopts a Finite State Machine (FSM) of three 32-bit words and two S-boxes to increase the resistance against the algebraic attacks by Billet and Gilbert [BG05]. Full evaluation of the design by the consortium is not public but a survey of this evaluation is given in [ETS06b]. Furthermore, SNOW 3G$^{\oplus}$ (in which the two modular additions are replaced by XORs) is defined and evaluated in [ETS06b]. The designers and external reviewers show that SNOW 3G has remarkable resistance against linear distinguishing attacks [NW06, WBdC03], while SNOW 3G$^{\oplus}$ offers much better resistance against algebraic attacks than SNOW 2.0$^{\oplus}$. A fault analysis of SNOW 3G is presented by Debraize [DC09] in 2009 using only 22 fault injections to reveal the secret key. In 2010 Brumley et al. [BHNS10] show a cache-timing attack on SNOW 3G retrieving the complete internal state from empirical timing data in a matter of seconds, without known keystream and only observation of a small number of cipher clocks.

The stream cipher SNOW 2.0 was published in 2002 by Ekdahl and Johansson [EJ02]. Previous attacks on SNOW 2.0 used linear distinguishers to distinguish the keystream of SNOW 2.0 from a truly random sequence. A distinguishing attack with linear masking was applied by Watanabe et al. [WBdC03] in 2003. It requires 2^{225} words of output and has a complexity of 2^{225}. This attack was later improved by Nyberg and Wallén [NW06] in 2006 to a linear distinguisher which needs 2^{179} bits of keystream and 2^{174} operations. A completely different kind of attack was proposed by Billet and Gilbert [BG05] in 2005. They showed that a slightly

simplified version of SNOW 2.0 can be broken using a linearization attack with a complexity of 2^{50} and only 1,000 words of keystream to get the initial state of the Linear Feedback Shift Register (LFSR). The extension of this attack to the actual SNOW 2.0 leads to a overdetermined system of quadratic equations for which the solving complexity remains unknown so far.

In this chapter we present the first attempt of cryptanalysis of SNOW 3G$^{\oplus}$ in public literature. We show that the feedback from the FSM to the LFSR during the initialization phase is vital for the security of this cipher, since we can break a version without such a feedback with two known IV's in $2^{56.1}$ time, 2^{33} data and 2^{25} memory complexity, and for any amount of initialization rounds! We then restore the feedback and study SNOW 3G$^{\oplus}$ against differential chosen IV attacks. We show attacks on SNOW 3G$^{\oplus}$ with 14, 15 and 16 rounds of initialization with time complexities $2^{42.1}$, $2^{91.3}$ and $2^{123.3}$ respectively. Afterwards, we perform the same kind of attacks on SNOW 2.0$^{\oplus}$. The known IV attack on a version without the feedback from the FSM to the LFSR during the initialization phase requires $2^{36.5}$ time, 62 data and 2^{30} memory complexities, and works for any amount of initialization rounds, too. For SNOW 2.0$^{\oplus}$ with the feedback we found differential chosen IV attacks on 14 up to 18 rounds of initialization with $2^{36.4}$ time and small memory complexities up to 2^{122} time and 2^{123} memory complexities.

This chapter is organized as follows: We give a description of SNOW 3G and SNOW 2.0 as well as their simplified versions SNOW 3G$^{\oplus}$ and SNOW 2.0$^{\oplus}$ in Section 5.1. The known and chosen IV attacks on SNOW 3G$^{\oplus}$ are presented in Section 5.2 and the similar attacks on SNOW 2.0$^{\oplus}$ are shown in Section 5.3. Finally, some conclusions are given in Section 5.4.

5.1 Description of both SNOW Ciphers

5.1.1 Description of SNOW 3G and SNOW 3G$^{\oplus}$

SNOW 3G is a word-oriented synchronous stream cipher with a 128-bit key and a 128-bit initial value (IV), each considered as four 32-bit words. It consists of a Linear Feedback Shift Register (LFSR) of sixteen 32-bit words and a Finite State Machine (FSM) with three 32-bit words, shown in Figure 5.1.

The symbol '$\oplus$' denotes the bitwise XOR and '$\boxplus$' denotes the addition modulo 2^{32}. The feedback word of the LFSR is recursively computed as

$$s_{15}^{t+1} = \alpha^{-1} \cdot s_{11}^t \oplus s_2^t \oplus \alpha \cdot s_0^t \ ,$$

where α is the root of the irreducible $GF(2^8)[x]$ polynomial

$$x^4 + \beta^{23}x^3 + \beta^{245}x^2 + \beta^{48}x + \beta^{239}$$

with β being the root of the irreducible $GF(2)[x]$ polynomial

$$x^8 + x^7 + x^5 + x^3 + 1 \ .$$

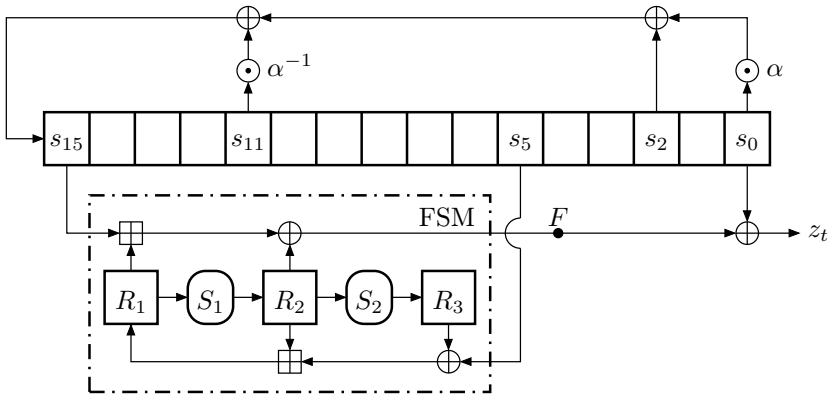

Figure 5.1: Keystream generation of SNOW 3G

The FSM has two input words s_{15}^t and s_5^t from the LFSR and computes at first the output word F as

$$F^t = (s_{15}^t \boxplus R_1^t) \oplus R_2^t \ .$$

Afterwards, the FSM is updated as

$$R_1^{t+1} = R_2^t \boxplus (R_3^t \oplus s_5^t) \qquad R_2^{t+1} = S_1(R_1^t) \qquad R_3^{t+1} = S_2(R_2^t) \ ,$$

where S_1 and S_2 are 32-bit to 32-bit S-boxes defined as compositions of four parallel applications of two 8-bit to 8-bit small S-boxes S_R and S_Q, with a linear diffusion layer respectively. Here S_R is the well-known AES S-box and S_Q is defined as

$$S_Q(y) = y \oplus y^9 \oplus y^{13} \oplus y^{15} \oplus y^{33} \oplus y^{41} \oplus y^{45} \oplus y^{47} \oplus y^{49} \oplus 0x25$$

for $y \in GF(2^8)$ defined by $x^8 + x^6 + x^5 + x^3 + 1$. If we decompose a 32-bit word w into four bytes $w = w^{(0)} \| w^{(1)} \| w^{(2)} \| w^{(3)}$ with $w^{(0)}$ being the most and $w^{(3)}$ the least significant bytes, then the S-boxes are

$$\begin{aligned} S_1(w) &= MC_1 \cdot \left[S_R(w^{(0)}), S_R(w^{(1)}), S_R(w^{(2)}), S_R(w^{(3)})\right]^T \\ S_2(w) &= MC_2 \cdot \left[S_Q(w^{(0)}), S_Q(w^{(1)}), S_Q(w^{(2)}), S_Q(w^{(3)})\right]^T \ , \end{aligned}$$

where MC_1 is the AES MixColumns operation for S_1 over $GF(2^8)$ defined by $x^8 + x^4 + x^3 + x + 1$ and MC_2 is the similar operation for S_2 over $GF(2^8)$ defined by $x^8 + x^6 + x^5 + x^3 + 1$.

SNOW 3G is initialized with the four word key $K = (k_0, k_1, k_2, k_3)$ and the four word $IV = (IV_0, IV_1, IV_2, IV_3)$. Let $\mathbf{1}$ be the all-one word, then the

LFSR is loaded as follows:

$$
\begin{array}{llll}
s_{15} = k_3 \oplus IV_0 & s_{14} = k_2 & s_{13} = k_1 & s_{12} = k_0 \oplus IV_1 \\
s_{11} = k_3 \oplus \mathbf{1} & s_{10} = k_2 \oplus \mathbf{1} \oplus IV_2 & s_9 = k_1 \oplus \mathbf{1} \oplus IV_3 & s_8 = k_0 \oplus \mathbf{1} \\
s_7 = k_3 & s_6 = k_2 & s_5 = k_1 & s_4 = k_0 \\
s_3 = k_3 \oplus \mathbf{1} & s_2 = k_2 \oplus \mathbf{1} & s_1 = k_1 \oplus \mathbf{1} & s_0 = k_0 \oplus \mathbf{1}
\end{array}
$$

The FSM is initialized with $R_1 = R_2 = R_3 = 0$. Then the cipher runs 32 clocks in the initialization mode doing:

1. compute the output F^t of the FSM,
2. update the FSM,
3. clock the LFSR with F^t XORed to the feedback of the LFSR.

After this, the cipher is switched to the keystream generation mode. As long as keystream is needed the cipher does:

1. compute the output F^t of the FSM,
2. update the FSM,
3. compute the keystream word $z^t = s_0^t \oplus F^t$,
4. clock the LFSR.

Please note that the first keystream word is discarded. Hence, there are 33 initialization rounds.

If we replace the two modulo additions in SNOW 3G by XORs we get SNOW 3G$^{\oplus}$. In each clock SNOW 3G$^{\oplus}$ executes two S-boxes, two multiplications and seven XORs. We will measure the time complexity of our attacks in SNOW 3G$^{\oplus}$ clocks.

5.1.2 Description of SNOW 2.0 and SNOW 2.0$^{\oplus}$

The only difference between SNOW 3G and SNOW 2.0 is the absence of the second S-box S_2 and the third word R_3 of the FSM as shown in Figure 5.2. All other equations, the initialization mode and the keystream generation are the same for SNOW 2.0. Two key sizes are allowed for SNOW 2.0 a 128-bit key and a 256-bit key but for the larger key the mixing and loading step is slightly different. A more detailed description is given in [EJ02]. In our attacks we concentrate on the 128-bit key version. If we replace the two modulo additions in SNOW 2.0 by XORs we get SNOW 2.0$^{\oplus}$. In each clock SNOW 2.0$^{\oplus}$ executes one S-box, two multiplications and six XORs. Analog to SNOW 3G$^{\oplus}$ we will measure the time complexity of our attacks on SNOW 2.0$^{\oplus}$ in clocks of SNOW 2.0$^{\oplus}$.

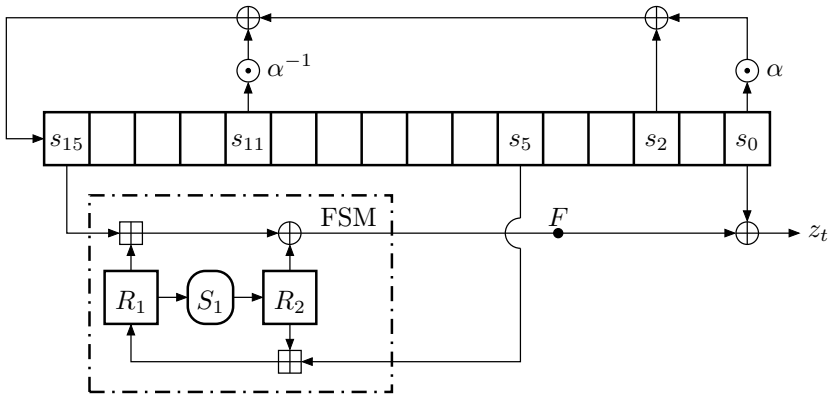

Figure 5.2: Figure of the keystream generation of SNOW 2.0

5.2 Known and Chosen IV Attacks on Versions of SNOW 3G$^{\oplus}$

In this section we will present a known IV attack on SNOW 3G$^{\oplus}$ without the FSM to LFSR feedback during the initialization phase. Afterwards, we restore the feedback and show a differential chosen IV attack on SNOW 3G$^{\oplus}$ with a reduced number of initialization rounds.

In both attacks the attacker has access to two keystreams corresponding to (K, IV_a) and (K, IV_b), where IV_a and IV_b are either arbitrary known IVs or IVs chosen by the attacker. The goal is to recover the internal state and thus the secret key from the known or chosen differences. All attacks have a similar structure, they begin with a **starting step**, which is different for each attack, followed by a **reduction step** and then finish with an **insertion step**.

In the **starting step** we only use the differences in the LFSR and the keystream to deduce potential differences in the FSM words exploiting the non-linearity from the S-boxes. In the **reduction step** we switch from the differences in the FSM words to the individual values of the FSM words to reduce the amount of potential differences exploiting the non-linearity of the S-boxes again. In the **insertion step** we simply insert the collected individual values for the FSM words in one system of keystream equations.

5.2.1 Difference Propagation through the S-boxes

To recover the initial state from the differences we need to know how the differences propagate through the S-boxes. Let $v_a, v_b, w_a, w_b, x_a, x_b$ be arbitrary

words satisfying the following equations for the first S-box

$$\begin{aligned} &\Delta v = v_a \oplus v_b \qquad w_a = S_1(v_a) \qquad w_b = S_1(v_b) \\ &\Delta w = w_a \oplus w_b = S_1(v_a) \oplus S_1(v_b) \ . \end{aligned}$$

We define a new notation for the last row

$$\overset{\text{out}}{\Delta}\, S_1(\Delta v) \overset{\text{def.}}{=} S_1(v_a) \oplus S_1(v_b) \ .$$

Similarly, we have for the second S-box

$$\begin{aligned} &\Delta w = w_a \oplus w_b \qquad x_a = S_2(w_a) \qquad x_b = S_2(w_b) \\ &\Delta x = x_a \oplus x_b = S_2(w_a) \oplus S_2(w_b) \ . \end{aligned}$$

Likewise, we define a new notation for the last row

$$\overset{\text{out}}{\Delta}\, S_2(\Delta w) \overset{\text{def.}}{=} S_2(w_a) \oplus S_2(w_b) \ .$$

In the same way we define the output difference for the small S-box S_R as $\overset{\text{out}}{\Delta}\, S_R(\Delta y)$ for a byte Δy and $\overset{\text{out}}{\Delta}\, S_Q(\Delta y)$ for the small S-box S_Q. For a fixed input difference Δy we get at most 127 possible output differences for the small S-box S_R. This yields approximately 2^{28} possible output differences for the S-box S_1 for a fixed input difference Δv. Also, we get at most 127 possible output differences for the small S-box S_Q resulting in about 2^{28} possible output differences for the S-box S_2. It is obvious that for a zero input difference, the output difference will be zero as well for all S-boxes. If we can fix the input difference Δv to a word with three bytes equal to zero and only one non-zero byte, we would get 2^7 possibilities for $\overset{\text{out}}{\Delta}\, S_1(\Delta v)$ as well as 2^7 possibilities for $\overset{\text{out}}{\Delta}\, S_2(\Delta v)$.

The knowledge of an input-output difference $\Delta v \to S_1 \to \Delta w$ with fixed differences Δv and Δw for the first S-box allows us to recover on average $(2 \cdot \frac{126}{127} + 4 \cdot \frac{1}{127})^4 \approx 16.51$ pairs of individual values for (v_a, v_b). If we know the pair (v_a, v_b) we know (w_a, w_b), too. The number of $2 \cdot \frac{126}{127} + 4 \cdot \frac{1}{127} \approx 2.02$ pairs comes from the difference propagation of the small S-box S_R. We have computed that the 256 pairs of individual values for each input difference unequal to zero yield exactly 127 output differences. In 126 cases the output difference is the same for two pairs and once four pairs result in the same output difference. Thus, in the 16.51 possible pairs of individual values we have around 8.255 times the case of two pairs (v_a, v_b) and (v_b, v_a).

Now we can apply the second S-box to the individual values w_a and w_b and yield values $x_a = S_2(w_a)$ and $x_b = S_2(w_b)$. The 16.51 possible pairs for the individual values result in 8.255 possibilities for Δx due to the fact that each two pairs (v_a, v_b) and (v_b, v_a) give two pairs (x_a, x_b) and (x_b, x_a) yielding the same difference Δx. If we can fix this difference as well, yielding a known input-output difference sequence $\Delta v \to S_1 \to \Delta w \to S_2 \to \Delta x$, only two pairs of individual values will conform to this difference sequence, namely (v_a, v_b) and (v_b, v_a).

5.2.2 Known IV Attack without FSM to LFSR Feedback

In this section we consider a known IV attack on SNOW 3G$^{\oplus}$ without the FSM to LFSR feedback, in which the attacker has access to two keystreams corresponding to (K, IV_a) and (K, IV_b), where IV_a and IV_b are arbitrary known IVs. This attack works for any number of initialization rounds.

During the keystream generation, we have the following equations for the differences at clock t:

$$\begin{aligned} \Delta z^t &= \Delta s_{15}^t \oplus \Delta R_1^t \oplus \Delta R_2^t \oplus \Delta s_0^t & \Delta R_2^t &= \overset{\text{out}}{\Delta} S_1(\Delta R_1^{t-1}) \\ \Delta R_1^t &= \Delta R_2^{t-1} \oplus \Delta R_3^{t-1} \oplus \Delta s_5^{t-1} & \Delta R_3^t &= \overset{\text{out}}{\Delta} S_2(\Delta R_2^{t-1}) \ . \end{aligned}$$

The differences in the LFSR depend on the known difference of the IVs and are completely predictable due to the linearity.

The main procedures of our attack are: Assume that $\Delta R_1^t = 0$ is given at time t. Deduce potential differences in the FSM words at different times from the linear evolution of the difference in the LFSR and the keystream difference equations. Knowing the input-output difference for the S-boxes, the few possibilities for the individual values of the FSM words are deduced. Combine the knowledge of the FSM state with that of the keystream to get linear equations on the LFSR state. Collect enough equations to get a solvable linear system which will recover the state of the LFSR. By the invertibility of the cipher run it backwards to find the 128-bit secret key K.

Assume $\Delta R_1^t = 0$. If this is not true, just take the next clock and so on. If we try this step 2^{32} times, then it will happen with a good probability. Denote the time when $\Delta R_1^t = 0$ by $t = 1$, then $\Delta R_1^1 = 0$ implicates $\Delta R_2^2 = 0$ and $\Delta R_3^3 = 0$ due to the difference propagation of the S-boxes. From the keystream equation at $t = 1$ we know ΔR_2^1. Similarly, we know ΔR_1^2 from which we can derive ΔR_3^1 via the update equation. Hereafter, we denote the known difference values by Δk_i and yield the state of the FSM as follows:

clock t	ΔR_1^t	ΔR_2^t	ΔR_3^t
1	0	Δk_0	Δk_2
2	Δk_1	0	
3			0

Starting step: For clock 3 we take the keystream equation and the update equation of ΔR_1^3, which are

$$\begin{aligned} \Delta z^3 &= \Delta s_{15}^3 \oplus \Delta R_1^3 \oplus \Delta R_2^3 \oplus \Delta s_0^3 \ , \\ \Delta R_1^3 &= \Delta R_2^2 \oplus \Delta R_3^2 \oplus \Delta s_5^2 \ , \end{aligned}$$

and combine them to one equation

$$\Delta R_3^2 \oplus \Delta R_2^3 = \Delta z^3 \oplus \Delta s_{15}^3 \oplus \Delta s_5^2 \oplus \Delta s_0^3 \ .$$

We know the right part completely and denote it with Δk_3. By the notations introduced in Section 5.2.1 we get

$$\overset{\text{out}}{\Delta} S_2(\Delta k_1) \oplus \overset{\text{out}}{\Delta} S_1(\Delta k_2) = \Delta k_3 \ . \tag{5.1}$$

We have $\frac{2^{28}.2^{28}}{2^{32}} = 2^{24}$ pairs satisfying this equation. To enumerate the possible pairs we first rewrite (5.1) as

$$\begin{pmatrix} \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(0)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(1)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(2)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(3)}) \end{pmatrix} = \begin{pmatrix} \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(0)}) \\ \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(1)}) \\ \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(2)}) \\ \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(3)}) \end{pmatrix} \oplus \begin{pmatrix} p_0^{\text{msb}} \\ p_1^{\text{msb}} \\ p_2^{\text{msb}} \\ p_3^{\text{msb}} \end{pmatrix} \oplus MC_1^{-1} . \begin{pmatrix} \Delta k_3^{(0)} \\ \Delta k_3^{(1)} \\ \Delta k_3^{(2)} \\ \Delta k_3^{(3)} \end{pmatrix},$$

where p_i^{msb} $(i = 0, 1, 2, 3)$ denotes a byte polynomial which contains only the most significant bits of all four values $\overset{\text{out}}{\Delta} S_Q$. For a detailed explanation please see Appendix B. Thus, we can fulfill the enumeration byte by byte. For the first row we need the value of $\overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(0)})$, which has 2^7 possibilities, and three more bits for p_0^{msb}. Then we check whether the value computed at the right side of the equation is a correct value for $\overset{\text{out}}{\Delta} S_R(\Delta k_2^{(0)})$. We will obtain 2^9 solutions for this byte equation. For the next three equations we already know the leading bits, which gives only 2^6 possibilities left in each byte equation resulting in 2^5 solutions. To get the solution of the word equation we have to combine the corresponding byte solutions and get $2^9 \cdot 2^5 \cdot 2^5 \cdot 2^5 = 2^{24}$ solutions, which needs about $2 \cdot 2^{24} = 2^{25}$ words of memory. The time complexity of this step is negligible as we only have a few XORs on bytes – the multiplication with MC_1^{-1} has to be done only once prior we try all the byte possibilities. Now the states of the FSM are as follows:

clock t	ΔR_1^t	ΔR_2^t	ΔR_3^t
1	0	Δk_0	Δk_2
2	Δk_1	0	(2^{24})
3	(2^{24})	(2^{24})	0
4			

Each possible value of ΔR_2^3 results in a possible value of ΔR_1^4. At clock 4 we have

$$\Delta R_2^3 \oplus \Delta R_2^4 = \Delta z^4 \oplus \Delta s_{15}^4 \oplus \Delta s_5^3 \oplus \Delta s_0^4 \ .$$

Replacing the difference ΔR_2^4 with the notation of Section 5.2.1 and denoting the known right part with Δk_4 we receive

$$\Delta R_2^3 \oplus \overset{\text{out}}{\Delta} S_1(\Delta R_1^3) = \Delta k_4 \ .$$

Let $\Delta R_1^3 = v^{(0)}\|v^{(1)}\|v^{(2)}\|v^{(3)}$ and $\Delta R_2^3 = w^{(0)}\|w^{(1)}\|w^{(2)}\|w^{(3)}$. Expanding this equation to the byte form we get

$$\begin{pmatrix} \overset{\text{out}}{\Delta} S_R(v^{(0)}) \\ \overset{\text{out}}{\Delta} S_R(v^{(1)}) \\ \overset{\text{out}}{\Delta} S_R(v^{(2)}) \\ \overset{\text{out}}{\Delta} S_R(v^{(3)}) \end{pmatrix} = MC_1^{-1} \cdot \begin{pmatrix} w^{(0)} \\ w^{(1)} \\ w^{(2)} \\ w^{(3)} \end{pmatrix} \oplus MC_1^{-1} \cdot \begin{pmatrix} \Delta k_4^{(0)} \\ \Delta k_4^{(1)} \\ \Delta k_4^{(2)} \\ \Delta k_4^{(3)} \end{pmatrix} .$$

We have to insert all 2^{24} possible pairs $(\Delta R_1^3, \Delta R_2^3)$ and verify the values $\overset{\text{out}}{\Delta} S_R$ for the single bytes. This results in a time complexity of $2^{24} \cdot \frac{1}{2} = 2^{23}$ clocks. There are $\frac{2^{24} \cdot 2^{28}}{2^{32}} = 2^{20}$ entries satisfying this equation. This means that we have 2^{20} sequences $(\Delta R_3^2, \Delta R_1^3, \Delta R_2^3, \Delta R_1^4, \Delta R_2^4)$ left which gives the following state of the FSM:

clock t	ΔR_1^t	ΔR_2^t	ΔR_3^t
1	0	Δk_0	Δk_2
2	Δk_1	0	(2^{20})
3	(2^{20})	(2^{20})	0
4	(2^{20})	(2^{20})	

The naive approach would be to continue checking the differences with the keystream equation at clock 5. The combined equation is

$$\Delta R_2^4 \oplus \overset{\text{out}}{\Delta} S_2(\Delta R_2^3) \oplus \overset{\text{out}}{\Delta} S_1(\Delta R_1^4) = \Delta k_5 \ .$$

We would have to insert all 2^{20} possibilities for the sequences and check two S-boxes. In total $\frac{2^{20} \cdot 2^{28} \cdot 2^{28}}{2^{32}} = 2^{44}$ sequences would satisfy this equation. Thus, we would get more and more sequences with each clock.

Instead of increasing the number of difference sequences we reduce them exploiting the non-linearity of the S-boxes. We do no longer consider only the differences of the FSM values. We rather switch to the individual values of the FSM words.

Reduction step: For each of the 2^{20} sequences we know the input-output difference of S_1 at clock 2 and 3: $\Delta R_1^2 \rightarrow S_1 \rightarrow \Delta R_2^3$. As explained in Section 5.2.1 we can recover 16.51 pairs $(R_{1,a}^2, R_{1,b}^2)$ on average. This means that we have 8.255 possible values for ΔR_3^4. Looking at clock 5 we have

$$\Delta R_2^4 \oplus \Delta R_3^4 \oplus \overset{\text{out}}{\Delta} S_1(\Delta R_1^4) = \Delta k_5 \ .$$

Let $\Delta R_1^4 = v^{(0)}\|v^{(1)}\|v^{(2)}\|v^{(3)}$, $\Delta R_2^4 = w^{(0)}\|w^{(1)}\|w^{(2)}\|w^{(3)}$ and $\Delta R_3^4 = x^{(0)}\|x^{(1)}\|x^{(2)}\|x^{(3)}$. We can rewrite this equation into byte form

$$\begin{pmatrix} \overset{\text{out}}{\Delta} S_R(v^{(0)}) \\ \overset{\text{out}}{\Delta} S_R(v^{(1)}) \\ \overset{\text{out}}{\Delta} S_R(v^{(2)}) \\ \overset{\text{out}}{\Delta} S_R(v^{(3)}) \end{pmatrix} = MC_1^{-1} \cdot \begin{pmatrix} w^{(0)} \\ w^{(1)} \\ w^{(2)} \\ w^{(3)} \end{pmatrix} \oplus MC_1^{-1} \cdot \begin{pmatrix} x^{(0)} \\ x^{(1)} \\ x^{(2)} \\ x^{(3)} \end{pmatrix} \oplus MC_1^{-1} \cdot \begin{pmatrix} \Delta k_5^{(0)} \\ \Delta k_5^{(1)} \\ \Delta k_5^{(2)} \\ \Delta k_5^{(3)} \end{pmatrix}.$$

We have to insert all 2^{20} possible pairs $(\Delta R_1^4, \Delta R_2^4)$ with the corresponding 8.255 possible values of ΔR_3^4 and verify the values $\overset{\text{out}}{\Delta} S_R$ for the single bytes. There are $\frac{2^{20} \cdot 8.255 \cdot 2^{28}}{2^{32}} \approx 2^{19.05}$ possible sequences left and the time complexity is $2^{20} \cdot 8.255 \cdot \frac{1}{2} \approx 2^{22.05}$ clocks. This identification of the individual values in the FSM for both keystreams has to be repeated for the next nine clocks to get enough individual values to compute the complete internal state. Each check will have a lower time complexity than the one before and will reduce the possible number of differences. The time complexity for all ten checks together in SNOW 3G$^{\oplus}$ clocks is

$$\left(\sum_{i=1}^{10} 2^{20} \cdot 8.255^i \cdot (2^{-4})^{i-1} \right) \cdot \frac{1}{2} \approx 2^{23.1} \quad .$$

During this reduction step the memory requirements will not exceed 2^{22} words. The number of sequences left is $2^{20} \cdot 8.255^{10} \cdot (2^{-4})^{10} \approx 2^{10.5}$.

Insertion step: Now we insert the individual values of the FSM into the keystream generation equations and the FSM update equations to get a linear system of the LFSR initial state. For each difference ΔR_1^i we have two pairs $(R_{1,a}^i, R_{1,b}^i)$ and $(R_{1,b}^i, R_{1,a}^i)$ for $i = 2, \ldots, 12$. Thus, we have to insert both values $R_{1,a}^i$ or $R_{1,b}^i$ but we only need the system of equations for one keystream. This would need a time complexity of $2^{10.5} \cdot 2^{10} \approx 2^{20.5}$ clocks. We can check with the keystream equation of clock 1 whether we took the right value of the pairs $(R_{1,a}^i, R_{1,b}^i)$. Then we clock backwards to receive the secret key K.

Results: The overall time complexity in SNOW 3G$^{\oplus}$ clocks is

$$2^{32} \cdot \left(2^{23} + 2^{23.1} + 2^{20.5}\right) \approx 2^{56.1} \quad .$$

The memory requirements are 2^{25} words and the keystream is of length 2^{33} words, especially 2^{32} consecutive words for both keystreams.

5.2.3 Differential Chosen IV Attacks

Now we look at SNOW 3G$^{\oplus}$ with the FSM to LFSR feedback during the initialization but reduced number of initialization rounds. We show attacks on 12 to 16 initialization rounds. All attacks are similar having the **starting step**, which is slightly different for each number of initialization rounds, followed by the **reduction step** and then finish with the **insertion step**. We consider a differential chosen IV attack scenario. Assume that we have two 128-bit IVs differing only in the most significant word IV_0, which gives the difference in s_{15} of the LFSR. As we will see below for the 13 and 14 rounds case we can restrict the difference to a single byte of IV_0 in order to reduce the complexity of our attacks. We denote this difference by Δd. Until round 11 this difference will not affect the FSM. After round 11 the known

Δd enters the FSM word R_1. In this section we denote the first displayed keystream word with z^0 and increase the clock for the following keystream words. The previous initialization rounds will have negative clocks. As described in Section 5.1.1 the first computed keystream word is discarded. Hence, clock -1 will have no feedback from the FSM to the LFSR and no keystream word.

Reduced Initialization of 12 Rounds.

Since all the differences in the FSM are zero, there are no differences fed back into the LFSR. Thus, the differences in the LFSR are all known. Our knowledge of the differences in the FSM is shown in the following table:

round	clock t	ΔR_1^t	ΔR_2^t	ΔR_3^t
12	-1	Δd	0	0
	0	Δd		0
	1			

The **starting step** is very easy because from the first keystream equation with $\Delta R_1^0 = \Delta d$ we get ΔR_2^0, which gives us immediately ΔR_1^1 and also ΔR_2^1 from the next keystream equation. Thus, we only have one known sequence $(\Delta R_1^{-1} = \Delta d,\ \Delta R_2^{-1} = \Delta R_3^{-1} = 0,\ \Delta R_1^0 = \Delta d,\ \Delta R_2^0,\ \Delta R_3^0 = 0,\ \Delta R_1^1,\ \Delta R_2^1)$.

We perform the **reduction step** in order to get the individual values for the words of the FSM. Starting with the known input-output difference $\Delta R_1^{-1} = \Delta d \rightarrow S_1 \rightarrow \Delta R_2^0$ we proceed as explained in the reduction step on page 57. The time complexity is $10 \cdot 8.255 \cdot \frac{1}{2} \approx 2^{5.4}$ clocks. We have one sequence left and for each ΔR_1^i only two pairs $(R_{1,a}^i, R_{1,b}^i)$ and $(R_{1,b}^i, R_{1,a}^i)$ with $i = -1, \ldots, 8$.

Afterwards, we execute the **insertion step** as explained on page 58 with a time complexity of 2^{10} clocks. We use the keystream equation of clock 12 to check the candidates.

Results: The total time complexity in SNOW 3G$^{\oplus}$ clocks is

$$2^{5.4} + 2^{10} \approx 2^{10.1} \ ,$$

the memory requirements are small and the needed keystream is only 12 consecutive words for each IV.

Reduced Initialization of 13 Rounds.

Here we extend the attack above by one more round. In the 13 round case, since until now all the differences in the FSM are either zero or the known Δd, no unknown difference was fed back into the LFSR. Thus, the differences

in the LFSR values are known. We know the differences in the FSM as shown in the following table:

round	clock t	ΔR_1^t	ΔR_2^t	ΔR_3^t
12	-2	Δd	0	0
13	-1	Δd		0
	0			

Starting step: From the first keystream equation and the update equations we receive

$$\Delta R_2^{-1} \oplus \Delta R_2^0 = \Delta z^0 \oplus \Delta s_{15}^0 \oplus \Delta s_5^{-1} \oplus \Delta s_0^0 \ .$$

Then we replace the differences at the left side with the notation introduced in Section 5.2.1, denote the known part at the right side with k_0 and obtain the equation

$$\overset{\text{out}}{\Delta} S_1(\Delta d) \oplus \overset{\text{out}}{\Delta} S_1(\Delta d) = \Delta k_0 \ .$$

Multiplying by MC_1^{-1} we get the byte form equation

$$\begin{pmatrix} \overset{\text{out}}{\Delta} S_R(\Delta d^{(0)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(1)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(2)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(3)}) \end{pmatrix} \oplus \begin{pmatrix} \overset{\text{out}}{\Delta} S_R(\Delta d^{(0)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(1)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(2)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(3)}) \end{pmatrix} = MC_1^{-1} \cdot \begin{pmatrix} \Delta k_0^{(0)} \\ \Delta k_0^{(1)} \\ \Delta k_0^{(2)} \\ \Delta k_0^{(3)} \end{pmatrix} \ .$$

We can check these four byte equations in a negligible time complexity as we only have a few XORs. The number of solutions will be $\frac{2^{28} \cdot 2^{28}}{2^{32}} = 2^{24}$ pairs of $(\Delta R_2^{-1}, \Delta R_2^0)$ which has memory requirements of 2^{25} words. We have 2^{24} sequences ($\Delta R_1^{-2} = \Delta d$, $\Delta R_2^{-2} = \Delta R_3^{-2} = 0$, $\Delta R_1^{-1} = \Delta d$, ΔR_2^{-1}, $\Delta R_3^{-1} = 0$, ΔR_1^0, ΔR_2^0).

Again, we go through the **reduction step** as explained on page 57 to reduce the number of sequences and to get the individual values for the FSM. We start with the input-output difference $\Delta R_1^{-2} = \Delta d \rightarrow S1 \rightarrow \Delta R_2^{-1}$. The time complexity of this step is $\left(\sum_{i=1}^{10} 2^{24} \cdot 8.255^i \cdot (2^{-4})^{i-1}\right) \cdot \frac{1}{2} \approx 2^{27.1}$ clocks. At the end we have $2^{24} \cdot 8.255^{10} \cdot (2^{-4})^{10} \approx 2^{14.5}$ difference sequences left. The memory requirements will not exceed 2^{26} words.

Now we perform the **insertion step** as explained on page 58, which would require a time complexity of $2^{14.5} \cdot 2^{10} \approx 2^{24.5}$ clocks.

Results: The overall time complexity in SNOW 3G$^{\oplus}$ clocks is

$$2^{27.1} + 2^{24.5} \approx 2^{27.3} \ .$$

The memory requirements are 2^{26} words and the keystream is of length 12 consecutive words for each IV.

Restriction: If we restrict the known arbitrary difference Δd to a word with three bytes equal to zero and only one non-zero byte, we can reduce our attack complexity considerably. After the **starting step** we only have one pair $(\Delta R_2^{-1}, \Delta R_2^0)$ of differences left, just as in the attack on 12 rounds explained on page 59. This way we will have the same time complexity of $2^{10.1}$ clocks and the memory requirements are small. The keystream will be of 12 consecutive words for each IV.

Reduced Initialization of 14 Rounds.

After 14 initialization rounds, we have one unknown difference in the LFSR since the difference ΔR_2^{-2} was fed back into the LFSR in round 13. All other differences, which were fed back, are either zero or the known Δd. We guess the individual value $R_{1,a}^{-3}$ for the first pair (K, IV_a) which has 2^{32} possibilities. From the value $R_{1,a}^{-3}$ we get with $\Delta R_1^{-3} = \Delta d$ the value $R_{1,b}^{-3}$ for the second pair (K, IV_b). Furthermore, we obtain the two pairs $(R_{2,a}^{-2}, R_{2,b}^{-2})$ and $(R_{3,a}^{-1}, R_{3,b}^{-1})$. We denote the now known difference ΔR_2^{-2} with Δk_0, the linear dependent ΔR_1^{-1} with Δk_1 and ΔR_3^{-1} with Δk_2. This way we can compute all differences of the LFSR because we know all differences coming from the FSM. We have the following knowledge of differences for the FSM:

round	clock t	ΔR_1^t	ΔR_2^t	ΔR_3^t
12	-3	Δd	0	0
13	-2	Δd	Δk_0	0
14	-1	Δk_1		Δk_2
	0			

Starting step: From the first keystream equation and the update equations for ΔR_1^0 and ΔR_2^0 we receive

$$\Delta R_2^{-1} \oplus \Delta R_2^0 = \Delta z^0 \oplus \Delta s_{15}^0 \oplus \Delta k_2 \oplus \Delta s_5^{-1} \oplus \Delta s_0^0 \ ,$$

which gives, with the notation of Section 5.2.1 and the known right part denoted as Δk_3, the equation

$$\overset{\text{out}}{\Delta} S_1(\Delta d) \oplus \overset{\text{out}}{\Delta} S_1(\Delta k_1) = \Delta k_3 \ .$$

We multiply the equation with MC_1^{-1} and rewrite it in byte notation as

$$\begin{pmatrix} \overset{\text{out}}{\Delta} S_R(\Delta d^{(0)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(1)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(2)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(3)}) \end{pmatrix} \oplus \begin{pmatrix} \overset{\text{out}}{\Delta} S_R(\Delta k_1^{(0)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_1^{(1)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_1^{(2)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_1^{(3)}) \end{pmatrix} = MC_1^{-1} \cdot \begin{pmatrix} \Delta k_3^{(0)} \\ \Delta k_3^{(1)} \\ \Delta k_3^{(2)} \\ \Delta k_3^{(3)} \end{pmatrix} \ .$$

Then we check this equation line by line for each byte in a negligible time complexity. The number of solutions will be $\frac{2^{28} \cdot 2^{28}}{2^{32}} = 2^{24}$ pairs $(\Delta R_2^{-1}, \Delta R_2^0)$ which need 2^{25} words of memory.

As expected we start the **reduction step** with the input-output difference $\Delta R_1^{-2} \rightarrow S_1 \rightarrow \Delta R_2^{-1}$. Since we start with 2^{24} sequences, we have exactly the same procedure as in the attack on 13 rounds on page 59 and thus the same complexities. The same applies to the **insertion step**.

Results: The overall time complexity is the same as in the 13 round case for each guess of $R_{1,a}^{-3}$, which gives

$$2^{32} \cdot \left(2^{27.1} + 2^{24.5}\right) \approx 2^{59.3}$$

SNOW 3G$^{\oplus}$ clocks. The memory requirements are 2^{26} words and the keystream is of length 12 words for each IV.

Restriction: If we restrict the known difference Δd to one non-zero byte only, we can reduce our attack complexity. We would start with guessing the individual value $R_{1,a}^{-3}$ for the first pair (K, IV_a) with 2^{32} possibilities, as explained above, and yield the same known differences of the FSM.

Then we proceed with the **starting step** as described. Due to the restriction only $\frac{2^7 \cdot 2^{28}}{2^{32}} = 2^3$ pairs $(\Delta R_2^{-1}, \Delta R_2^0)$ satisfy the equation, which require 2^4 words of memory.

During the **reduction step** the amount of sequences decreases to one. We start with the input-output difference $\Delta R_1^{-2} \rightarrow S_1 \rightarrow \Delta R_2^{-1}$ and get the individual values for 10 clocks in total. The time complexity is $2^{6.5}$ clocks and the memory will not exceed 2^6 words.

The **insertion step** is the same as in the 12 round case explained on page 59 with a time complexity of 2^{10} clocks.

Results: The total time complexity for this attack with the restriction is

$$2^{32} \cdot \left(2^{6.5} + 2^{10}\right) \approx 2^{42.1}$$

SNOW 3G$^{\oplus}$ clocks. The memory requirements are 2^6 words and the keystream is of length 12 words for each IV.

Reduced Initialization of 15 Rounds and 16 Rounds.

In the 15 round case two unknown differences ΔR_2^{-3} and ΔR_2^{-2} were fed back into the LFSR. We guess the individual values of $R_{1,a}^{-4}$ and $R_{1,a}^{-3}$ for the first pair (K, IV_a) with complexity of 2^{64}. From the value $R_{1,a}^{-4}$ and $\Delta R_1^{-4} = \Delta d$ we get the value of $R_{1,b}^{-4}$ and following the pairs $(R_{2,a}^{-3}, R_{2,b}^{-3})$ and $(R_{3,a}^{-2}, R_{3,b}^{-2})$. Denote the known differences ΔR_2^{-3} by Δk_0, ΔR_1^{-2} by Δk_1 and ΔR_3^{-2} by Δk_2. From $R_{1,a}^{-3}$ and $\Delta R_1^{-3} = \Delta d$ we get the value of $R_{1,b}^{-3}$ and following the pairs $(R_{2,a}^{-2}, R_{2,b}^{-2})$ and $(R_{3,a}^{-1}, R_{3,b}^{-1})$. Again, we denote the now known

differences ΔR_2^{-2} by Δk_3, ΔR_1^{-1} by Δk_4 and ΔR_3^{-1} by Δk_5. This gives the following differences for the FSM:

round	clock t	ΔR_1^t	ΔR_2^t	ΔR_3^t
12	-4	Δd	0	0
13	-3	Δd	Δk_0	0
14	-2	Δk_1	Δk_3	Δk_2
15	-1	Δk_4		Δk_5
	0			

We now have the same starting point as in the attack on 14 initialization rounds described on page 61. For all three steps we proceed in the same way as explained there.

Results: Since we guessed one more word in the beginning of the attack the time complexity becomes

$$2^{32} \cdot 2^{59.3} \approx 2^{91.3}$$

SNOW 3G$^{\oplus}$ clocks. The memory and keystream requirements remain.

A **restriction** of the Δd is no longer useful because the value $\overset{\text{out}}{\Delta} S_1(\Delta d)$ is not computed anymore.

In the 16 round case we guess one more word and then proceed as in the attack on 15 rounds. The time complexity in SNOW 3G$^{\oplus}$ clocks is

$$2^{32} \cdot 2^{91.3} \approx 2^{123.3}$$

and the memory and keystream requirements remain.

The summary of our results is given in Table 5.1 with time in SNOW 3G$^{\oplus}$ clocks and keystream and memory in words.

5.3 Known and Chosen IV Attacks on Versions of SNOW 2.0$^{\oplus}$

In this section we will show a known IV attack on SNOW 2.0$^{\oplus}$ without the FSM to LFSR feedback during the initialization phase. Afterwards, we restore the feedback and present a chosen IV attack on SNOW 2.0$^{\oplus}$ with a reduced number of initialization rounds. Both attacks are similar to the attacks on SNOW 3G$^{\oplus}$ described in Section 5.2.

In both attacks on SNOW 2.0$^{\oplus}$ the attacker has access to two keystreams corresponding to (K, IV_a) and (K, IV_b), where IV_a and IV_b are either arbitrary known IVs or IVs chosen by the attacker. The goal is to recover the internal state and thus the secret key from the known or chosen differences.

target	keystream	time	memory
SNOW 3G$^{\oplus}$ without feedback	2^{33}	$2^{56.1}$	2^{25}
SNOW 3G$^{\oplus}$ with feedback			
12 initialization rounds	24	$2^{10.1}$	small
13 initialization rounds	24	$2^{27.3}$	2^{26}
with restriction of Δd to 1 byte	24	$2^{10.1}$	small
14 initialization rounds	24	$2^{59.3}$	2^{26}
with restriction of Δd to 1 byte	24	$2^{42.1}$	2^{6}
15 initialization rounds	24	$2^{91.3}$	2^{26}
16 initialization rounds	24	$2^{123.3}$	2^{26}

Table 5.1: Summary of our known and chosen IV attacks on SNOW 3G$^{\oplus}$

For the propagation of the differences through the S-box we refer to Section 5.2.1 considering only the first S-box there.

All attacks have a similar structure, they begin with a **starting step**, which is different for each attack, followed by a **reduction step** and then finish with an **insertion step**.

In the **starting step** we either know or guess the first difference occurring in the FSM and deduce the potential differences for the FSM words using the next keystream equation. In the **reduction step** we reduce the amount of possible differences to one using the keystream and update equations for the next clocks always performing the same check. In the **insertion step** we compute the individual values for the differences in the FSM words, insert them into one system of keystream equations and check for the right insertion.

5.3.1 Known IV Attack without FSM to LFSR Feedback

In this section we consider a known IV attack on SNOW 2.0$^{\oplus}$ without the FSM to LFSR feedback, in which the attacker has access to two keystreams corresponding to (K, IV_a) and (K, IV_b), where IV_a and IV_b are arbitrary known IVs. This attack works for any number of initialization rounds.

During the keystream generation we have the following equations for the differences at clock t:

$$\begin{aligned} \Delta z^t &= \Delta s_{15}^t \oplus \Delta R_1^t \oplus \Delta R_2^t \oplus \Delta s_0^t \\ \Delta R_1^t &= \Delta R_2^{t-1} \oplus \Delta s_5^{t-1} \\ \Delta R_2^t &= \overset{\text{out}}{\Delta} S_1(\Delta R_1^{t-1}) \ . \end{aligned}$$

The differences in the LFSR depend on the known difference of the IVs and are completely predictable due to the linearity.

The main procedures of our attack are: Guess ΔR_1^t at time t. Deduce potential differences in the FSM words at different times from the linear evolution of the difference in the LFSR and the keystream difference equations. Knowing the input-output difference for the S-box, the few possibilities for the individual values of the FSM words are deduced. Combine the knowledge of the FSM state with that of the keystream to get linear equations on the LFSR state. Collect enough equations to get a solvable linear system which will recover the state of the LFSR. By the invertibility of the cipher run it backwards to find the 128-bit secret key K.

Starting step: We guess at an arbitrary clock $t = 1$ the value of ΔR_1^1 which gives us directly ΔR_2^1 via the keystream equation of clock 1. Furthermore, we get ΔR_1^2 via the update equation of ΔR_2^1 and with the next keystream equation ΔR_2^2. On the other hand, we know $\Delta R_2^2 = \overset{\text{out}}{\Delta}\, S_1(\Delta R_1^1)$ as explained in Section 5.2.1. Thus, we can check our guess for ΔR_1^1 immediately by combining the keystream equations of clock 1 and 2 with the update equations for ΔR_1^2 and ΔR_2^2 yielding

$$\overset{\text{out}}{\Delta}\, S_1(\Delta R_1^1) \oplus \Delta R_1^1 = \Delta z^2 \oplus \Delta s_{15}^2 \oplus \Delta z^1 \oplus \Delta s_{15}^1 \oplus \Delta s_0^1 \oplus \Delta s_5^1 \oplus \Delta s_0^2 \ .$$

We know the right part of this equation completely and denote it with Δk_0. Let $\Delta R_1^1 = v^{(0)} \| v^{(1)} \| v^{(2)} \| v^{(3)}$. We multiply the whole equation with MC_1^{-1} and receive it in byte form

$$\begin{pmatrix} \overset{\text{out}}{\Delta}\, S_R(v^{(0)}) \\ \overset{\text{out}}{\Delta}\, S_R(v^{(1)}) \\ \overset{\text{out}}{\Delta}\, S_R(v^{(2)}) \\ \overset{\text{out}}{\Delta}\, S_R(v^{(3)}) \end{pmatrix} = MC_1^{-1} \cdot \begin{pmatrix} v^{(0)} \\ v^{(1)} \\ v^{(2)} \\ v^{(3)} \end{pmatrix} \oplus MC_1^{-1} \cdot \begin{pmatrix} \Delta k_0^{(0)} \\ \Delta k_0^{(1)} \\ \Delta k_0^{(2)} \\ \Delta k_0^{(3)} \end{pmatrix} \ .$$

The values $\overset{\text{out}}{\Delta}\, S_R$ for the small S-box can be checked bytewise. We have to insert all 2^{32} possibilities for ΔR_1^1 which results in a time complexity of 2^{32} clocks. For each ΔR_1^1 we have 2^{28} possibilities for $\Delta R_2^2 = \overset{\text{out}}{\Delta}\, S_1(\Delta R_1^1)$ and therefore $\frac{2^{32} \cdot 2^{28}}{2^{32}} = 2^{28}$ pairs $(\Delta R_1^1, \Delta R_2^2)$ remain after the check. The memory requirements to store these pairs are 2^{29} words. With the linear dependencies of the differences we have 2^{28} sequences $(\Delta R_1^1, \Delta R_2^1, \Delta R_1^2, \Delta R_2^2)$.

Reduction step: For clock 3 we take the keystream equation and the update equations for the FSM and obtain

$$\Delta R_2^2 \oplus \Delta R_2^3 = \Delta z^3 \oplus \Delta s_{15}^3 \oplus \Delta s_5^2 \oplus \Delta s_0^3 \ .$$

Denoting the known right part with Δk_1 and using the notation of Section 5.2.1 we get

$$\Delta R_2^2 \oplus \overset{\text{out}}{\Delta}\, S_1(\Delta R_1^2) = \Delta k_1 \ .$$

Let $\Delta R_1^2 = w^{(0)} \| w^{(1)} \| w^{(2)} \| w^{(3)}$ and $\Delta R_2^2 = x^{(0)} \| x^{(1)} \| x^{(2)} \| x^{(3)}$. We can rewrite this equation into byte form

$$\begin{pmatrix} \overset{\text{out}}{\Delta} S_R(w^{(0)}) \\ \overset{\text{out}}{\Delta} S_R(w^{(1)}) \\ \overset{\text{out}}{\Delta} S_R(w^{(2)}) \\ \overset{\text{out}}{\Delta} S_R(w^{(3)}) \end{pmatrix} = MC_1^{-1} \cdot \begin{pmatrix} x^{(0)} \\ x^{(1)} \\ x^{(2)} \\ x^{(3)} \end{pmatrix} \oplus MC_1^{-1} \cdot \begin{pmatrix} \Delta k_1^{(0)} \\ \Delta k_1^{(1)} \\ \Delta k_1^{(2)} \\ \Delta k_1^{(3)} \end{pmatrix} .$$

We have to insert all 2^{28} possibilities for $(\Delta R_1^2, \Delta R_2^2)$, which requires a time complexity of 2^{28} clocks, and can check the values for $\overset{\text{out}}{\Delta} S_R$ as explained before. The amount of $\frac{2^{28} \cdot 2^{28}}{2^{32}} = 2^{24}$ triples $(\Delta R_1^2, \Delta R_2^2, \Delta R_2^3)$ remains after the check. With the linear dependencies of the differences we have 2^{24} sequences $(\Delta R_1^1, \Delta R_2^1, \Delta R_1^2, \Delta R_2^2, \Delta R_1^3, \Delta R_2^3)$. We can repeat this check until we only have one sequence left. Since every check will reduce the number of possibilities as well as the time complexity for the next check by 2^{-4}, we will need six more clocks. The time complexity for all seven checks together is $\sum_{i=0}^{6} 2^{28} \cdot 2^{-4i} \approx 2^{28.1}$. At the end, we only have one sequence $(\Delta R_1^i, \Delta R_2^i)$ left with $i = 1, \ldots, 9$. For this sequence we need to compute $(\Delta R_1^{10}, \Delta R_2^{10})$.

Insertion step: Now we can switch from the differences to the individual values for $(R_1^i, R_2^{i+1}), i = 1, \ldots, 9$. Each fixed input-output difference $\Delta R_1^i \rightarrow S_1 \rightarrow \Delta R_2^{i+1}$ has about 16.51 pairs $(R_{1,a}^i, R_{1,b}^i)$ with $i = 1, \ldots, 9$ as explained in Section 5.2.1. We need the system of equations for one keystream only, and insert always the first value of each pair. The clocks 1 to 9 are sufficient to get 16 consecutive values for the LFSR. Altogether, we get $16.51^9 \approx 2^{36.4}$ possible insertions. Thus, we use 22 consecutive keystream equations starting with clock 10 to verify which of the $2^{36.4}$ possible insertions was correct. Consequently, this insertion step has a time complexity of $2^{36.4}$ clocks.

Results: In summary, our attack will have a time complexity of

$$2^{32} + 2^{32.1} + 2^{36.4} \approx 2^{36.5}$$

SNOW $2.0^{\oplus}$ clocks. We need 2^{29} words of memory and $9 + 22 = 31$ consecutive words for both keystreams.

5.3.2 Differential Chosen IV Attacks

Now we look at SNOW $2.0^{\oplus}$ with the FSM to LFSR feedback during the initialization but reduced number of initialization rounds. We show attacks on 12 to 18 initialization rounds. All attacks are similar having the **starting step**, which is slightly different for each number of initialization rounds, followed by the **reduction step** and then finish with the **insertion step**. We consider a differential chosen IV attack scenario. Assume that we have two 128-bit IVs differing only in the most significant word IV_0, which gives the difference in s_{15} of the LFSR. We denote the difference in the IV word

IV_0 by Δd. As we will see below we can restrict this difference to a single byte of Δd in order to reduce the complexity of our attacks. Until round 11 the difference will not affect the FSM. After round 11 the known Δd enters the FSM word R_1. In this section we denote the first displayed keystream word with z^0 and increase the clock for the following keystream words. The previous initialization rounds will have negative clocks. As described in Section 5.1.2 the first computed keystream word is discarded. Hence, clock -1 will have no feedback from the FSM to the LFSR and no keystream word.

Reduced Initialization of 12 Rounds.

The differences in the LFSR values are known because so far all differences in the FSM are zero and therefore no unknown difference was fed back into the LFSR.

In the **starting step** we know $\Delta R_1^0 = \Delta d$ for the first keystream equation. Therefore, we know ΔR_2^0 and consequentially ΔR_1^t and ΔR_2^t for all clocks.

We do not need the **reduction step** because we already have only one sequence of differences. We simply compute all differences up to $(\Delta R_1^{10}, \Delta R_2^{10})$ via the update and keystream equations. Then we perform the **insertion step** of the individual values for R_1 and R_2 as explained in the attack in Section 5.3.1.

Results: Our attack has a time complexity of $2^{36.4}$ SNOW 2.0$^{\oplus}$ clocks, memory requirements are small and we need 31 consecutive words for both keystreams.

Reduced Initialization of 13 Rounds.

We extend the attack above by one more round. All differences in the FSM are zero or the known Δd until now. Therefore, no unknown difference was fed back into the LFSR and all differences of the LFSR values are known. Our knowledge of the differences in the FSM is shown in the following table:

round	clock t	ΔR_1^t	ΔR_2^t
12	-2	Δd	0
13	-1	Δd	
	0		

Starting step: From the first keystream equation and the update equations for ΔR_1^0 and ΔR_2^0 with notation of Section 5.2.1 we get the equation

$$\overset{\text{out}}{\Delta} S_1(\Delta d) \oplus \overset{\text{out}}{\Delta} S_1(\Delta d) = \Delta z^0 \oplus \Delta s_{15}^0 \oplus \Delta s_5^{-1} \oplus \Delta s_0^0 \ .$$

We denote the known part at the right side of this equation as Δk_0, multiply the whole equation with MC_1^{-1} and gain the byte form equation

$$\begin{pmatrix} \overset{\text{out}}{\Delta} S_R(\Delta d^{(0)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(1)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(2)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(3)}) \end{pmatrix} \oplus \begin{pmatrix} \overset{\text{out}}{\Delta} S_R(\Delta d^{(0)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(1)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(2)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta d^{(3)}) \end{pmatrix} = MC_1^{-1} \cdot \begin{pmatrix} \Delta k_0^{(0)} \\ \Delta k_0^{(1)} \\ \Delta k_0^{(2)} \\ \Delta k_0^{(3)} \end{pmatrix} .$$

We can check these four byte equations in a negligible time complexity as we only have a few XORs. The number of solutions will be $\frac{2^{28} \cdot 2^{28}}{2^{32}} = 2^{24}$ pairs $(\Delta R_2^{-1}, \Delta R_2^0)$ requiring 2^{25} words of memory.

Reduction step: We start the check with the keystream and update equations for clock 1 and have to insert all 2^{24} pairs $(\Delta R_1^0, \Delta R_2^0)$ similarly to the reduction step in Section 5.3.1. Then we proceed for the next five clocks until we only have one sequence $(\Delta R_1^i, \Delta R_2^i)$ left with $i = -1, \ldots, 6$. For this sequence we compute the differences up to $(\Delta R_1^8, \Delta R_2^8)$ via the update and keystream equations. The time complexity for all six checks together is $\sum_{i=0}^{5} 2^{24} \cdot 2^{-4i} \approx 2^{24.1}$.

Insertion step: Our starting point for the computation of the individual values for the FSM is the input-output difference $\Delta R_1^{-1} \rightarrow S_1 \rightarrow \Delta R_2^0$. Then we proceed as explained in the insertion step in Section 5.3.1.

Results: Altogether, the time complexity in SNOW $2.0^{\oplus}$ clocks is

$$2^{24.1} + 2^{36.4} \approx 2^{36.4} \quad .$$

The memory requirements are 2^{25} words and we need 31 consecutive words for both keystreams.

Restriction: If we would restrict the known difference Δd to a word which has three bytes equal to zero and only one non-zero byte, we would get only one pair after the **starting step**. Thus, we have no **reduction step** and the same **insertion step** as in the attack on 12 rounds described on page 67. There would be no influence on the time complexity and the needed keystream because the insertion step is the expensive one, but we could reduce the memory requirements to a small amount.

Reduced Initialization of 14 Rounds.

After 14 initialization rounds we have one unknown difference in the LFSR. All other differences in the LFSR values are either zero or the known Δd. We guess this unknown difference $\Delta R_2^{-2} = \overset{\text{out}}{\Delta} S_1(\Delta d)$ which has 2^{28} possibilities.

We denote the now known ΔR_2^{-2} with Δk_0 and the linear dependent ΔR_1^{-1} with Δk_1. This gives the following differences for the FSM:

round	clock t	ΔR_1^t	ΔR_2^t
12	-3	Δd	0
13	-2	Δd	k_0
14	-1	k_1	
	0		

With this knowledge of the FSM differences we are at the same starting point as in the attack on 13 rounds on page 67.

Results: Thus, we would have the same memory and keystream requirements and the time complexity is

$$2^{28} \cdot 2^{36.4} \approx 2^{64.4}$$

SNOW 2.0$^{\oplus}$ clocks due to the extra guess.

Consideration with more memory: If we consider the 2^{28} extra possibilities immediately in the **starting step** we would get 2^{52} pairs $(\Delta R_2^{-1}, \Delta R_2^0)$ which would need 2^{53} words of memory. During the **reduction step** we decrease this amount to one using 13 checks until clock 14. Afterwards, we perform the **insertion step** with this single sequence. Then the time complexity for the starting step is 2^{37}, for the reduction step it is $2^{52.1}$, and for the insertion step it remains. This concludes to a total time complexity of $2^{52.1}$ clocks. The memory requirements are 2^{53} words and the needed keystream remains.

Restriction: With the restriction of Δd to a word with only one non-zero byte we would have 2^3 pairs $(\Delta R_2^{-1}, \Delta R_2^0)$ after the **starting step**. Thus, the time complexity for the starting step and the reduction step is negligible, and for the insertion step it remains. This concludes to a total time complexity of $2^{36.4}$ SNOW 2.0$^{\oplus}$ clocks. The memory requirements are small and the needed keystream remains.

Reduced Initialization up to 18 Rounds.

We can pursue these attacks for the next four rounds until initialization round 18. Prior to the starting step we would have to guess all unknown differences which were fed back into the LFSR. Then we are always at the same starting point as described in the 14 round case before. The **restriction** of Δd to a word with only one non-zero byte reduces the complexities in all these cases. The **consideration** with more memory, as explained in the 14 round case before, is applicable with or without the restriction for all cases.

The summary of our results is given in Table 5.2 with time in SNOW $2.0^{\oplus}$ clocks and keystream and memory in words. If one of the requirements exceeds 2^{128} the attempt is not considered as a promising attack anymore.

target	keystream	time	memory
SNOW $2.0^{\oplus}$ without feedback	62	$2^{36.5}$	2^{30}
SNOW $2.0^{\oplus}$ with feedback			
12 initialization rounds	62	$2^{36.4}$	small
13 initialization rounds	62	$2^{36.4}$	2^{25}
with restriction of Δd to 1 byte	62	$2^{36.4}$	small
14 initialization rounds	62	$2^{64.4}$	2^{25}
with more memory	62	$2^{52.1}$	2^{53}
with restriction of Δd to 1 byte	62	$2^{36.4}$	small
15 initialization rounds	62	$2^{92.4}$	2^{25}
with more memory	62	$2^{80.1}$	2^{81}
with restriction of Δd to 1 byte	62	$2^{50.4}$	2^{25}
with restriction and more memory	62	$2^{38.4}$	2^{39}
16 initialization rounds			
with more memory	62	$2^{108.1}$	2^{109}
with restriction of Δd to 1 byte	62	$2^{78.4}$	2^{25}
with restriction and more memory	62	2^{66}	2^{67}
17 initialization rounds			
with restriction of Δd to 1 byte	62	$2^{106.4}$	2^{25}
with restriction and more memory	62	2^{94}	2^{95}
18 initialization rounds			
with restriction and more memory	62	2^{122}	2^{123}

Table 5.2: Summary of our known and chosen IV attacks on SNOW $2.0^{\oplus}$

5.4 Summary

In the first half of this chapter we have shown known IV and chosen IV resynchronization attacks on SNOW $3G^{\oplus}$. We can attack any amount of initialization rounds of SNOW $3G^{\oplus}$ if there is no feedback from FSM to LFSR. With such feedback we show key-recovery attacks up to 16 rounds of initialization and use only a few keystream words. Our results indicate that about half of the initialization rounds of SNOW $3G^{\oplus}$ might succumb to chosen IV resynchronization attacks. However, the remaining security margin is quite significant and thus these attacks pose no threat to the security of SNOW 3G itself. In the second half we have shown known IV and chosen IV resynchronization attacks on SNOW $2.0^{\oplus}$. Likewise, we can attack

any amount of initialization rounds of SNOW 2.0$^{\oplus}$ if there is no feedback from FSM to LFSR. With such feedback we are able to mount key-recovery attacks up to 18 rounds of initialization using only a few keystream words. This shows that more than half of the initialization rounds of SNOW 2.0$^{\oplus}$ succumb to chosen IV resynchronization attacks. However, the remaining security margin is big enough and thus these attacks pose no threat to the security of SNOW 2.0 itself.

Chapter 6

Attacks on Simplified Versions of K2

The stream cipher K2 was proposed by Kiyomoto, Tanaka and Sakurai at SECRYPT 2007 [KTS07]. Currently it is submitted to the International Organization for Standardization (ISO) for standardization.

A security evaluation is given in [BPR11] with the conclusion that no weaknesses were found. The attack idea of [BHNS10] on SNOW 3G is applicable to K2 as well. Some side-channel attacks are applied on K2 in [HYY+10] showing that K2 offers reasonable resistance to side-channel attacks even without countermeasures.

In this chapter we present two attacks on simplified versions of K2. We show a differential chosen IV attack with key-recovery on simplified versions with five, six and seven initialization clocks with time complexity of $2^{10.05}$, $2^{18.05}$ and $2^{47.05}$ clocks. The needed keystream amounts to 28 words for the attacks on five and six initialization clocks and 24 words for the attack on seven initialization clocks. For a simplified version with seven initialization clocks we show a distinguishing attack with a time complexity of $2^{27.6}$ clocks, needed keystream of 2^{32} words and negligible memory requirements.

This chapter is organized as follows. We give a description of the cipher K2 and its simplification K2$^{\oplus}$ in Section 6.1. The differential chosen IV attack with key-recovery on K2$^{\oplus}$ with five, six and seven initialization clocks is presented in Section 6.2. In Section 6.3 we describe the distinguishing attack on K2$^{\oplus}$ with seven initialization clocks. Some conclusions are given in Section 6.4.

6.1 Description of K2 and K2$^{\oplus}$

Kiyomoto, Tanaka and Sakurai proposed the stream cipher K2 at SECRYPT 2007 [KTS07]. It consists of two Feedback Shift Registers (FSR), a Dynamic Feedback Controller (DFC), which dynamically chooses between four

different feedback functions, and a Non-Linear Function (NLF) as shown in Figure 6.1.

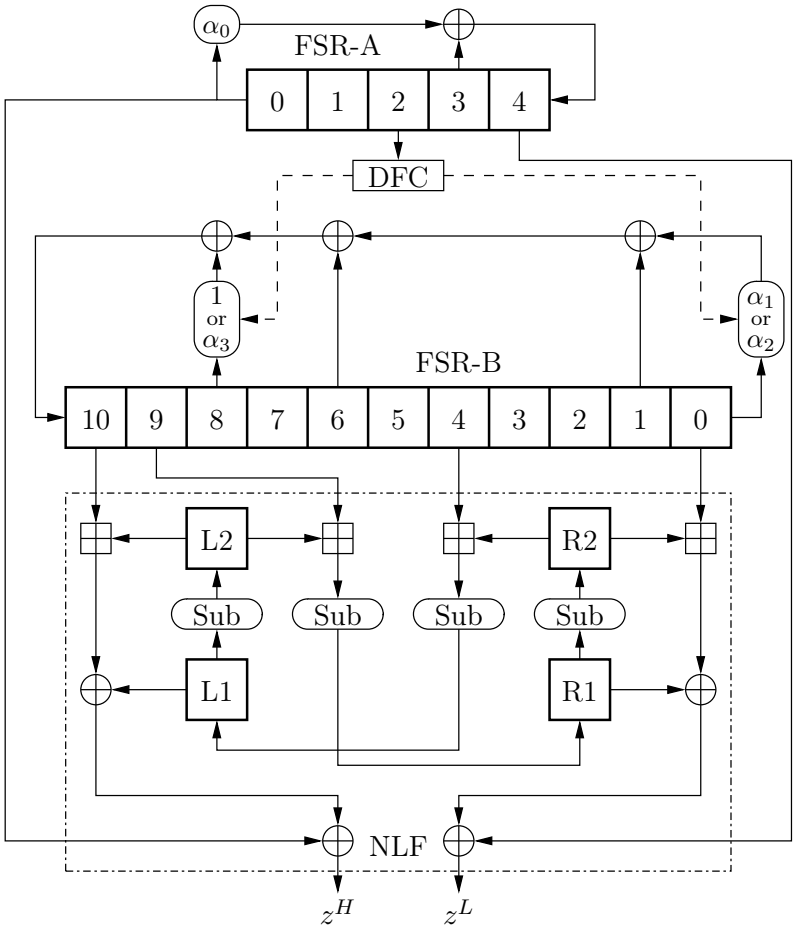

Figure 6.1: Keystream generation of K2

The first FSR, called $FSR-A$, has five words $(a_4, \ldots, a_0)$ each of size 32 bit. The feedback function is

$$a_4^t = \alpha_0 a_0^{t-1} \oplus a_3^{t-1}$$

where the multiplier α_0 is a constant chosen as the root of an irreducible polynomial of fourth degree in $GF(2^8)[x]$. The second FSR, called $FSR-B$, has eleven words $(b_{10}, \ldots, b_0)$ each of size 32 bit. The feedback function of $FSR-B$ is selected by the DFC. This controller has two clock control bits $cl1$ and $cl2$ which are described as

$$cl1_t = a_2^t[30] \quad \text{and} \quad cl2_t = a_2^t[31]$$

with $a_2^t[30]$ being the second most significant bit (smsb) of a_2^t and $a_2^t[31]$ being the most significant bit (msb) of a_2^t. Then the feedback function of the $FSR-B$ is

$$b_{10}^t = (\alpha_1^{cl1_{t-1}} + \alpha_2^{1-cl1_{t-1}} - 1)b_0^{t-1} \oplus b_1^{t-1} \oplus b_6^{t-1} \oplus \alpha_3^{cl2_{t-1}} b_8^{t-1}$$

where the multipliers $\alpha_{(1,2,3)}$ are constants each one chosen as the root of a different irreducible polynomial of fourth degree in $GF(2^8)[x]$. The NLF has four words memory $(L1, L2, R1, R2)$ and four times a Sub function. This Sub function operates on a word and uses the 8-bit AES S-box [DR02] and the AES MixColumns operation over $GF(2^8)$ defined by $x^8+x^4+x^3+x+1$. The exact work flow for a word $w = w^{(0)}\|w^{(1)}\|w^{(2)}\|w^{(3)}$ with $w^{(0)}$ being the most and $w^{(3)}$ being the least significant bytes is

$$Sub(w) = MC \cdot \left[S(w^{(0)}), S(w^{(1)}), S(w^{(2)}), S(w^{(3)})\right]^T$$

with S denoting the AES S-box and MC the MixColumns operation. With each clock the output of the NLF is the keystream of two words (z_t^H, z_t^L) computed as

$$\begin{aligned} z_t^H &= (b_{10}^t \boxplus L2^t) \oplus L1^t \oplus a_0^t \\ z_t^L &= (b_0^t \boxplus R2^t) \oplus R1^t \oplus a_4^t \ . \end{aligned}$$

The symbol '$\oplus$' denotes the bitwise XOR and the symbol '$\boxplus$' denotes the addition modulo 2^{32}. To update the memory words of the NLF compute

$$\begin{aligned} L1^t &= Sub(R2^{t-1} \boxplus b_4^{t-1}) \qquad L2^t = Sub(L1^{t-1}) \\ R1^t &= Sub(L2^{t-1} \boxplus b_9^{t-1}) \qquad R2^t = Sub(R1^{t-1}) \ . \end{aligned}$$

The K2 cipher uses a four word key $K = [K_0, K_1, K_2, K_3]$ and a four word $IV = [IV_0, IV_1, IV_2, IV_3]$. For the loading step two intermediate results are computed using the Sub function

$$\begin{aligned} S1 &= Sub[(K_3 \ll 8) \oplus (K_3 \gg 24)] \oplus 0x01000000 \\ S2 &= Sub[((K_0 \oplus K_1 \oplus K_2 \oplus K_3 \oplus S1) \ll 8) \\ &\quad \oplus ((K_0 \oplus K_1 \oplus K_2 \oplus K_3 \oplus S1) \gg 24)] \oplus 0x02000000 \ , \end{aligned}$$

the constants at the end are given in hexadecimal numbers. Afterwards, the FSRs are loaded

$$\begin{aligned} a_0 &= K_0 \oplus S1 & b_3 &= IV_1 \\ a_1 &= K_3 & b_4 &= K_0 \oplus S1 \oplus S2 \\ a_2 &= K_2 & b_5 &= K_1 \oplus S2 \\ a_3 &= K_1 & b_6 &= IV_2 \\ a_4 &= K_0 & b_7 &= IV_3 \\ b_0 &= K_0 \oplus K_2 \oplus S1 \oplus S2 & b_8 &= K_0 \oplus K_1 \oplus K_2 \oplus K_3 \oplus S1 \\ b_1 &= K_1 \oplus K_3 \oplus S2 & b_9 &= K_0 \oplus K_1 \oplus S1 \\ b_2 &= IV_0 & b_{10} &= K_0 \oplus K_1 \oplus K_2 \oplus S1 \ . \end{aligned}$$

The four memory words of the NLF are initialized with zero. Then, during the initialization the cipher is clocked 24 times doing

1. get the output from the NLF (z_t^H, z_t^L),
2. update the NLF,
3. update $FSR-B$ with z_t^H is XORed to the new word of $FSR-B$,
4. update $FSR-A$ with z_t^L is XORed to the new word of $FSR-A$.

After this initialization the K2 cipher produces the keystream and the FSRs are updated without feedback from the NLF.

For the rest of this chapter we consider a simplified version of K2 where all additions modulo 2^{32} are replaced with XOR and denote this version with K2$^{\oplus}$.

We measure the time complexity for our attacks in K2$^{\oplus}$ clocks.

6.2 Differential Chosen IV Attack with Key-Recovery

Considering a differential chosen IV scenario, we choose two different IVs IV_a and IV_b. We know the keystream from both pairs (K, IV_a) and (K, IV_b) with unknown key K. Both IVs only differ in IV word IV_1 which takes the longest until it enters the NLF. We denote this chosen difference with Δd. The DFC always takes the msb and smsb of a_2. Thus, we do not want to have a difference there. Accordingly, we choose the starting difference Δd with msb and smsb equal to zero. Our goal is to recover the whole internal state right after the loading step which means we get the unknown key K.

We denote the first displayed keystream words with z_0^H and z_0^L and increase the clock for the following keystream words. The previous initialization clocks will have negative clocks.

6.2.1 Difference Propagation through the S-boxes

From the differences in the keystream and the partially known differences in the FSRs we compute the differences in the words of the NLF. We then need to know how the differences in the NLF words propagate through the Sub function. Let v_a, v_b, w_a, w_b be arbitrary words at different times t with the following equations:

$$\begin{aligned} &\Delta v = v_a \oplus v_b \qquad w_a = Sub(v_a) \qquad w_b = Sub(v_b) \\ &\Delta w = w_a \oplus w_b = Sub(v_a) \oplus Sub(v_b) \ . \end{aligned}$$

We define a new notation

$$\overset{\text{out}}{\Delta} Sub(\Delta v) \overset{\text{def.}}{=} Sub(v_a) \oplus Sub(v_b) \ .$$

In the same way we define a new notation for the AES S-box S. Let x_a, x_b, y_a, y_b be arbitrary bytes at different times t with the following equations:

$$\begin{aligned} &\Delta x = x_a \oplus x_b \qquad y_a = S(x_a) \qquad y_b = S(x_b) \\ &\Delta y = y_a \oplus y_b = S(x_a) \oplus S(x_b) \ . \end{aligned}$$

Then the new notation is

$$\overset{\text{out}}{\Delta} S(\Delta x) \overset{\text{def.}}{=} S(x_a) \oplus S(x_b) \ .$$

During the keystream generation we have the following equations for the differences at clock t:

$$\begin{aligned} \Delta z_t^H &= \Delta b_{10}^t \oplus \Delta L1^t \oplus \Delta L2^t \oplus \Delta a_0^t \\ \Delta z_t^L &= \Delta b_0^t \oplus \Delta R1^t \oplus \Delta R2^t \oplus \Delta a_4^t \end{aligned}$$

$$\begin{aligned} \Delta L1^t &\overset{\text{out}}{=} \Delta Sub\,(\Delta R2^{t-1} \oplus \Delta b_4^{t-1}) & \Delta R1^t &\overset{\text{out}}{=} \Delta Sub\,(\Delta L2^{t-1} \oplus \Delta b_9^{t-1}) \\ \Delta L2^t &\overset{\text{out}}{=} \Delta Sub\,(\Delta L1^{t-1}) & \Delta R2^t &\overset{\text{out}}{=} \Delta Sub\,(\Delta R1^{t-1}) \ . \end{aligned}$$

We define two notations for the XORed sums in the Sub functions at time $t-1$

$$\Delta Rb^{t-1} \overset{\text{def.}}{=} (\Delta R2^{t-1} \oplus \Delta b_4^{t-1}) \quad \text{and} \quad \Delta Lb^{t-1} \overset{\text{def.}}{=} (\Delta L2^{t-1} \oplus \Delta b_9^{t-1}) \ .$$

Any fixed input difference $\Delta Rb^{t-1} \neq 0$ results in nearly 2^{28} possible output differences for $\Delta L1^t$, because any input difference in the small 8-bit AES S-box S results in 127 possible output differences. If we know or fix the input-output difference of Sub meaning $\Delta Rb^{t-1} \rightarrow Sub \rightarrow \Delta L1^t$, we can recover on average $(2 \cdot \frac{126}{127} + 4 \cdot \frac{1}{127})^4 \approx 16.51$ pairs of individual values (Rb_a^{t-1}, Rb_b^{t-1}) for Sub. Due to the difference propagation in the small S-box S, we have $2 \cdot \frac{126}{127} + 4 \cdot \frac{1}{127} \approx 2.02$ pairs on byte level. From the pair (Rb_a^{t-1}, Rb_b^{t-1}) we obtain $(L1_a^t, L1_b^t)$ and $(L2_a^{t+1}, L2_b^{t+1})$, too, after applying the Sub function once and twice. This means that we have $\frac{16.51}{2} \approx 8.255$ possible values for $\Delta L2^{t+1}$. If we know or fix this difference as well, we have only two pairs of individual values left which satisfy the sequence $\Delta Rb^{t-1} \rightarrow Sub \rightarrow \Delta L1^t \rightarrow Sub \rightarrow \Delta L2^{t+1}$, namely (Rb_a^{t-1}, Rb_b^{t-1}) and (Rb_b^{t-1}, Rb_a^{t-1}).

For a fixed input-output difference $\Delta Lb^{t-1} \rightarrow Sub \rightarrow \Delta R1^t$ we can recover on average 16.51 pairs of individual values (Lb_a^{t-1}, Lb_b^{t-1}) for Sub in the same way as described above resulting in 8.255 possible values for $\Delta R2^{t+1}$. Similarly, with this difference known or fixed we have only two pairs of individual values remaining which satisfy the sequence $\Delta Lb^{t-1} \rightarrow Sub \rightarrow \Delta R1^t \rightarrow Sub \rightarrow \Delta R2^{t+1}$, namely (Lb_a^{t-1}, Lb_b^{t-1}) and (Lb_b^{t-1}, Lb_a^{t-1}).

If we have collected enough individual values for the NLF, we can derive some words for $FSR-B$ from the update equations $L1^t = Sub(R2^{t-1} \oplus b_4^{t-1})$ and $R1^t = Sub(L2^{t-1} \oplus b_9^{t-1})$ of the NLF. The insertion of the individual values of the NLF together with some words of $FSR-B$ in the keystream equations yields some words for $FSR-A$. At the end, we know enough words of the NLF, $FSR-B$ and $FSR-A$ to clock backwards and reveal the secret key.

6.2.2 Reduced Initialization of 5 Clocks

We reduce the number of initialization clocks of K2$^{\oplus}$ to five. In the 5$^{\text{th}}$ initialization clock the starting difference Δd enters the word $R1$ of the NLF. During the keystream computation there is no more feedback from the NLF to the FSRs anymore. Therefore, we know all differences of $FSR{-}A$ as Δd enters it in clock 4 and then propagates linearly. For $FSR{-}B$ the computation of the differences only depends on the unknown bits of the DFC which select the multipliers for the $FSR{-}B$ feedback.

The propagation of the chosen difference Δd during the five initialization clocks is shown in Table 6.1. We omit the zero differences and write a question mark for unknown differences. The work flow of K2 has two steps, first computing the keystream words, then updating the internal state. The first row in Table 6.1 only shows the starting state. Then each row shows the updated state of K2$^{\oplus}$. Thus, the first keystream words are computed from the values in the last row of this table.

clock	FSR-B											FSR-A					NLF			
ini	10	9	8	7	6	5	4	3	2	1	0	4	3	2	1	0	L1	L2	R1	R2
								d												
1									d											
2										d										
3	d										d									
4	?	d										d								
5	?	?	d									d	d						?	

Table 6.1: Propagation of the difference d through K2$^{\oplus}$ during the five initialization clocks

The differences of the keystream words for clock 0 are

$$\begin{aligned} \Delta z_0^H &= \Delta b_{10}^0 \oplus \Delta L1^0 \oplus \Delta L2^0 \oplus \Delta a_0^0 \\ \Delta z_0^L &= \Delta b_0^0 \oplus \Delta R1^0 \oplus \Delta R2^0 \oplus \Delta a_4^0 \ , \end{aligned}$$

where the differences in $\Delta L1^0, \Delta L2^0, \Delta a_0^0, \Delta b_0^0, \Delta R2^0$ are all equal to zero and $\Delta a_4^0 = \Delta d$. Hence, we know Δb_{10}^0 and $\Delta R1^0$. The update equation of $FSR{-}B$ implies $\Delta b_{10}^0 = \Delta b_{10}^{-1} = \Delta b_9^0$ and $\Delta b_{10}^0 = \Delta b_9^1$. Furthermore, we know $\Delta Lb^{-1} = \Delta L2^{t-1} \oplus \Delta b_9^{t-1} = 0 \oplus \Delta d$ and $\Delta R1^0$ which fixes the input-output difference $\Delta Lb^{-1} \rightarrow Sub \rightarrow \Delta R1^0$. As explained in Section 6.2.1 we can recover on average 16.51 pairs of individual values for Sub resulting in 8.255 possible values for $\Delta R2^1$.

Clock 1 gives us $\Delta z_1^H = \Delta b_{10}^1 = \Delta b_9^2$ because all other differences are zero. For Δz_1^L we can rewrite the keystream equation in the following way:

$$\begin{aligned} & & \Delta z_1^L &= \Delta b_0^1 \oplus \Delta R1^1 \oplus \Delta R2^1 \oplus \Delta a_4^1 \\ \Leftrightarrow & & \Delta R1^1 &= \Delta b_0^1 \oplus \Delta R2^1 \oplus \Delta a_4^1 \oplus \Delta z_1^L \\ \Leftrightarrow & \overset{\text{out}}{\Delta}\, Sub(\Delta L2^0 \oplus \Delta b_9^0) &= 0 \oplus \Delta R2^1 \oplus \Delta d \oplus \Delta z_1^L \\ \Leftrightarrow & \overset{\text{out}}{\Delta}\, Sub(0 \oplus \Delta b_9^0) &= \Delta R2^1 \oplus \Delta k_0 \ , \end{aligned}$$

so that we know all differences at the right side with $\Delta k_0 = \Delta d \oplus \Delta z_1^L$. Let $\Delta b_9^0 = v^{(0)} \| v^{(1)} \| v^{(2)} \| v^{(3)}$ and $\Delta R2^1 = w^{(0)} \| w^{(1)} \| w^{(2)} \| w^{(3)}$. We can undo the MixColumns and yield the equation in byte form

$$\begin{pmatrix} \overset{\text{out}}{\Delta}\, S(v^{(0)}) \\ \overset{\text{out}}{\Delta}\, S(v^{(1)}) \\ \overset{\text{out}}{\Delta}\, S(v^{(2)}) \\ \overset{\text{out}}{\Delta}\, S(v^{(3)}) \end{pmatrix} = MC_1^{-1} \cdot \begin{pmatrix} w^{(0)} \\ w^{(1)} \\ w^{(2)} \\ w^{(3)} \end{pmatrix} \oplus MC_1^{-1} \cdot \begin{pmatrix} \Delta k_0^{(0)} \\ \Delta k_0^{(1)} \\ \Delta k_0^{(2)} \\ \Delta k_0^{(3)} \end{pmatrix} \ .$$

Then we insert all 8.255 possibilities of $\Delta R2^1$ and check byte by byte whether the computed value on the right side is a valid difference for the left side. The time complexity for this check is two clocks and only one pair $(\Delta R1^1, \Delta R2^1)$ will remain due to $\frac{2^{28} \cdot 8.255}{2^{32}} < 1$. The known difference $\Delta R2^1$ fixes the difference sequence $\Delta Lb^{-1} \rightarrow Sub \rightarrow \Delta R1^0 \rightarrow Sub \rightarrow \Delta R2^1$ leaving only two pairs of individual values satisfying it. Furthermore, we know $\Delta Lb^0 = \Delta L2^0 \oplus \Delta b_9^0 = 0 \oplus \Delta d$ and $\Delta R1^1$ which fixes the input-output difference $\Delta Lb^0 \rightarrow Sub \rightarrow \Delta R1^1$. Thus, we can recover nearly 16.51 pairs of individual values following 8.255 possible differences for $\Delta R2^2$.

In clock 2 Δb_{10}^2 has two possibilities because in its update equation

$$\Delta b_{10}^2 = (\alpha_1^{cl1_1} + \alpha_2^{1-cl1_1} - 1)\Delta b_0^1 \oplus \Delta b_1^1 \oplus \Delta b_6^1 \oplus \alpha_3^{cl2_1} \Delta b_8^1 \ ,$$

all differences are equal to zero except for Δb_8^1. The DFC takes $cl2_1$ to select the multiplier. We have a zero in the msb of $\Delta a_2^1 = \Delta d$ and following $\Delta b_{10}^2 = \Delta b_8^1$ or $\Delta b_{10}^2 = \alpha_3 \Delta b_8^1$. We can rewrite the keystream equation in the following way:

$$\begin{aligned} & & \Delta z_2^H &= \Delta b_{10}^2 \oplus \Delta L1^2 \oplus \Delta L2^2 \oplus \Delta a_0^2 \\ \Leftrightarrow & & \Delta L1^2 &= \Delta b_{10}^2 \oplus \Delta L2^2 \oplus \Delta a_0^2 \oplus \Delta z_2^H \\ \Leftrightarrow & \overset{\text{out}}{\Delta}\, Sub(\Delta R2^1 \oplus \Delta b_4^1) &= \Delta b_{10}^2 \oplus 0 \oplus 0 \oplus \Delta z_2^H \\ \Leftrightarrow & \overset{\text{out}}{\Delta}\, Sub(\Delta R2^1 \oplus 0) &= \Delta b_{10}^2 \oplus \Delta z_2^H \ , \end{aligned}$$

so that we know all differences at the right side. We insert both possibilities for Δb_{10}^2, undo the MixColumns operation and check bytewise whether the computed value on the right side is a valid difference for the left side. The

time complexity for this check is a half clock and only one pair $(\Delta b_{10}^2, \Delta L1^2)$ will remain. We know $\Delta Rb^1 = \Delta R2^1 \oplus \Delta b_4^1 = \Delta R2^1 \oplus 0$ and $\Delta L1^2$ which fixes the input-output difference $\Delta Rb^1 \rightarrow Sub \rightarrow \Delta L1^2$. Therefore, we have nearly 16.51 pairs of individual values following 8.255 possibilities for $\Delta L2^3$. For Δz_2^L we do exactly the same as described for Δz_1^L at clock 1 receiving known values for $\Delta R1^1$ and $\Delta R2^2$ as well as 8.255 possibilities for $\Delta R2^3$.

For clock 3 we have two possibilities for Δb_{10}^3 and 8.255 possibilities for $\Delta L2^3$. Thus, we rewrite the equation

$$\begin{aligned} & \Delta z_3^H &=& \Delta b_{10}^3 \oplus \Delta L1^3 \oplus \Delta L2^3 \oplus \Delta a_0^3 \\ \Leftrightarrow \quad & \Delta L1^3 &=& \Delta b_{10}^3 \oplus \Delta L2^3 \oplus \Delta a_0^3 \oplus \Delta z_3^H \\ \Leftrightarrow \quad & \overset{\text{out}}{\Delta}\, Sub(\Delta R2^2 \oplus \Delta b_4^2) &=& \Delta b_{10}^3 \oplus \Delta L2^3 \oplus \Delta d \oplus \Delta z_3^H \\ \Leftrightarrow \quad & \overset{\text{out}}{\Delta}\, Sub(\Delta R2^2 \oplus 0) &=& \Delta b_{10}^3 \oplus \Delta L2^3 \oplus \Delta d \oplus \Delta z_3^H \ , \end{aligned}$$

insert all listed possibilities, undo the MixColumns operation and check byte by byte. The time complexity for this check is four clocks and only one triple $(\Delta b_{10}^3, \Delta L1^3, \Delta L2^3)$ will remain due to $\frac{2^{28} \cdot 2 \cdot 8.255}{2^{32}} \approx 1$. Furthermore, we know $\Delta Rb^2 = \Delta R2^2 \oplus \Delta b_4^2 = \Delta R2^2 \oplus 0$ and $\Delta L1^3$ which fixes the input-output difference $\Delta Rb^2 \rightarrow Sub \rightarrow \Delta L1^3$. Thus, we can recover 8.255 possibilities for $\Delta L2^4$. In addition, the known difference $\Delta L2^3$ fixes the difference sequence $\Delta Rb^1 \rightarrow Sub \rightarrow \Delta L1^2 \rightarrow Sub \rightarrow \Delta L2^3$ yielding only two pairs of individual values satisfying it. For Δz_3^L we do exactly the same as described for Δz_1^L at clock 1 receiving known values for $\Delta R1^2$ and $\Delta R2^3$ as well as 8.255 possibilities for $\Delta R2^4$.

In clock 4, 5 and 6 we do exactly the same as described for clock 3.

Now we have collected enough individual values for the NLF. So far, the time complexity is $2 + \frac{1}{2} + 2 + 4 \cdot (4 + 2) = 28.5$ clocks.

For each key-IV pair we have a system of keystream and update equations. We select one of them for the insertion of these individual values. Since we collected 20 pairs of individual values, always two pairs for one difference, we take the first value of a pair to be inserted in the system of equations. Wrong allocations will have contradictions somewhere in the equations or at least in the keystream for clock 7 and are dispelled this way. This would result in a time complexity of 2^{10} clocks. For the right allocation at clock 6 we can clock backwards and receive the secret key with time complexity of six clocks.

The overall time complexity in K2$^{\oplus}$ clocks is

$$28.5 + 2^{10} + 6 \approx 2^{10.05} \ .$$

The needed keystream amounts to two words per clock for seven clocks for each pair (K, IV_a) and (K, IV_b) which yields 28 keystream words. The memory requirements are roughly 30 words.

6.2.3 Reduced Initialization of 6 Clocks

We extend the previous attack by one more clock. In the 5^{th} initialization clock the starting difference Δd enters the word $R1$ of the NLF resulting in an unknown difference $\Delta R1^{-1}$, which is fed back into $FSR{-}A$. We guess $\Delta R1^{-1}$ with $\Delta R1^{-1} \overset{\text{out}}{=\!\!=} Sub(\Delta L2^{-2} \oplus \Delta b_9^{-2}) \overset{\text{out}}{=\!\!=} Sub(0 \oplus \Delta d)$. Thus, we know the input difference and have 2^{28} possibilities for the output difference $\Delta R1^{-1}$. We can only employ those $\Delta R1^{-1}$ which have msb and smsb equal to zero because otherwise we will have differences in the bits for the DFC. This reduces the possibilities for $\Delta R1^{-1}$ to 2^{26} but we have to try the whole attack 2^2 times to let the case msb=smsb=0 happen. We denote the now known $\Delta R1^{-1}$ with Δk_0.

During the keystream computation there is no more feedback from the NLF to the FSRs anymore. Therefore, we know all differences of $FSR{-}A$ as Δd enters it in clock 4, Δk_0 enters it in clock 6 and both propagate linearly. For $FSR{-}B$ the computation of the differences only depends on the unknown bits of the DFC which select the multipliers for the $FSR{-}B$ feedback. We have to guess the difference $\Delta b_{10}^{-2} = \Delta b_{10}^{-1}$ because we need this difference for the update equations of the NLF as well as for the feedback function of $FSR{-}B$. The difference $\Delta b_{10}^{-2} = \Delta b_{10}^{-1}$ has only two possibilities due to the fact that in its update equation all differences are zero except for $\Delta b_0^{-3} = \Delta d$. For this difference we need to guess the choice of the multiplier which depends on the smsb of a_2^{-3}. We denote the now known $\Delta b_{10}^{-2} = \Delta b_{10}^{-1}$ with Δk_1.

Table 6.2 shows the propagation of Δd and all guessed differences. Zero differences are omitted and unknown differences are noted with question marks.

clock	FSR-B											FSR-A					NLF			
ini	10	9	8	7	6	5	4	3	2	1	0	4	3	2	1	0	L1	L2	R1	R2
								d												
1									d											
2										d										
3	d										d									
4	k_1	d										d								
5	k_1	k_1	d									d	d					k_0		
6	?	k_1	k_1	d								k_0	d	d				?	?	

Table 6.2: Propagation of the differences $\Delta d, \Delta k_0$ and Δk_1 through $\text{K2}^{\oplus}$ during the six initialization clocks

The differences of the keystream words for clock 0 are

$$\begin{aligned} \Delta z_0^H &= \Delta b_{10}^0 \oplus \Delta L1^0 \oplus \Delta L2^0 \oplus \Delta a_0^0 \\ \Delta z_0^L &= \Delta b_0^0 \oplus \Delta R1^0 \oplus \Delta R2^0 \oplus \Delta a_4^0 \ , \end{aligned}$$

where the differences in $\Delta L1^0$, $\Delta L2^0$, Δa_0^0, Δb_0^0 are zero, and $\Delta a_4^0 = \Delta k_0$. Hence, we know Δb_{10}^0. As $\Delta R1^{-1} = \Delta k_0$ is fixed we know the input-output difference $\Delta Lb^{-2} \rightarrow Sub \rightarrow \Delta R1^{-1} = \Delta k_0$. Thus, we can recover on average 16.51 pairs of individual values for this sequence resulting in 8.255 possibilities for $\Delta R2^0$. For Δz_0^L we can rewrite the keystream equation in the following way:

$$\begin{aligned} & \Delta z_0^L &=& \Delta b_0^0 \oplus \Delta R1^0 \oplus \Delta R2^0 \oplus \Delta a_4^0 \\ \Leftrightarrow \quad & \Delta R1^0 &=& \Delta b_0^0 \oplus \Delta R2^0 \oplus \Delta a_4^0 \oplus \Delta z_0^L \\ \Leftrightarrow \quad & \overset{\text{out}}{\Delta}\, Sub(\Delta L2^{-1} \oplus \Delta b_9^{-1}) &=& 0 \oplus \Delta R2^0 \oplus \Delta k_0 \oplus \Delta z_0^L \\ \Leftrightarrow \quad & \overset{\text{out}}{\Delta}\, Sub(0 \oplus \Delta k_1) &=& \Delta R2^0 \oplus \Delta k_0 \oplus \Delta z_0^L \ . \end{aligned}$$

Then we can insert all 8.255 possibilities for $\Delta R2^0$, undo the MixColumns operation and check bytewise whether the value on the right side is a valid difference for the left side. The time complexity for this check is two clocks and only one pair $(\Delta R1^0, \Delta R2^0)$ will survive due to $\frac{2^{28} \cdot 8.255}{2^{32}} < 1$. The $\Delta R2^0$ leaves only two pairs of individual values which fulfills the sequence $\Delta Lb^{-2} \rightarrow Sub \rightarrow \Delta R1^{-1} = \Delta k_0 \rightarrow Sub \rightarrow \Delta R2^0$. The $\Delta R1^0$ fixes the input-output difference $\Delta Lb^{-1} \rightarrow Sub \rightarrow \Delta R1^0$ resulting in nearly 16.51 pairs of individual values following 8.255 possible differences for $\Delta R2^1$.

In clock 1 Δb_{10}^1 has two possibilities because in its update equation

$$\Delta b_{10}^1 = (\alpha_1^{cl1_0} + \alpha_2^{1-cl1_0} - 1)\Delta b_0^0 \oplus \Delta b_1^0 \oplus \Delta b_6^0 \oplus \alpha_3^{cl2_0} \Delta b_8^0 \ ,$$

all differences are equal to zero except for Δb_8^0. Hence, we have $\Delta b_{10}^1 = \Delta b_8^0$ or $\Delta b_{10}^1 = \alpha_3 \Delta b_8^0$ because we have a zero in the msb of $\Delta a_2^0 = \Delta d$. We can rewrite the keystream equation in the following way:

$$\begin{aligned} & \Delta z_1^H &=& \Delta b_{10}^1 \oplus \Delta L1^1 \oplus \Delta L2^1 \oplus \Delta a_0^1 \\ \Leftrightarrow \quad & \Delta L1^1 &=& \Delta b_{10}^1 \oplus \Delta L2^1 \oplus \Delta a_0^1 \oplus \Delta z_1^H \\ \Leftrightarrow \quad & \overset{\text{out}}{\Delta}\, Sub(\Delta R2^0 \oplus \Delta b_4^0) &=& \Delta b_{10}^1 \oplus 0 \oplus \Delta d \oplus \Delta z_1^H \\ \Leftrightarrow \quad & \overset{\text{out}}{\Delta}\, Sub(\Delta R2^0 \oplus 0) &=& \Delta b_{10}^1 \oplus \Delta d \oplus \Delta z_1^H \ , \end{aligned}$$

so that we know all differences at the right side. We insert both possibilities for Δb_{10}^1, undo the MixColumns operation and check bytewise whether the value on the right side is a valid one for the left side. The time complexity for this check is a half clock and only one pair $(\Delta b_{10}^1, \Delta L1^1)$ will remain. The $\Delta L1^1$ fixes the input-output difference $\Delta Rb^0 \rightarrow Sub \rightarrow \Delta L1^1$ which results in nearly 16.51 pairs of individual values following 8.255 possibilities for $\Delta L2^2$. For Δz_1^L we do exactly the same as described for Δz_0^L at clock 0.

For clock 2 we have two possibilities for Δb_{10}^2 and 8.255 possibilities for $\Delta L2^2$. Thus, we rewrite the equation

$$\begin{aligned} & \Delta z_2^H &=& \Delta b_{10}^2 \oplus \Delta L1^2 \oplus \Delta L2^2 \oplus \Delta a_0^2 \\ \Leftrightarrow \quad & \Delta L1^2 &=& \Delta b_{10}^2 \oplus \Delta L2^2 \oplus \Delta a_0^2 \oplus \Delta z_2^H \\ \Leftrightarrow \quad & \overset{\text{out}}{\Delta}\, Sub(\Delta R2^1 \oplus \Delta b_4^1) &=& \Delta b_{10}^2 \oplus \Delta L2^2 \oplus \Delta d \oplus \Delta z_2^H \\ \Leftrightarrow \quad & \overset{\text{out}}{\Delta}\, Sub(\Delta R2^1 \oplus 0) &=& \Delta b_{10}^2 \oplus \Delta L2^2 \oplus \Delta d \oplus \Delta z_2^H \ , \end{aligned}$$

insert all listed possibilities, undo the MixColumns operation and check byte by byte. The time complexity for this check is four clocks and only one triple $(\Delta b_{10}^2, \Delta L1^2, \Delta L2^2)$ will remain due to $\frac{2^{28}\cdot 2\cdot 8.255}{2^{32}} \approx 1$. The difference $\Delta L1^2$ fixes the sequence $\Delta Rb^1 \rightarrow Sub \rightarrow \Delta L1^2$ which results in 8.255 possibilities for $\Delta L2^3$. The known difference $\Delta L2^2$ fixes the pair of individual values for the sequence $\Delta Rb^0 \rightarrow Sub \rightarrow \Delta L1^1 \rightarrow Sub \rightarrow \Delta L2^2$. For Δz_2^L we do exactly the same as described for Δz_0^L at clock 0.

For the clocks 3, 4 and 5 we do exactly the same as described for clock 2.

Now we have collected enough individual values for the NLF. So far, the time complexity is $2+\frac{1}{2}+2+4\cdot(4+2) = 28.5$ clocks. It is the same as for the computations for five initialization clocks as we executed similar equations.

For each key-IV pair we have a system of keystream and update equations. We select one of them for the insertion of these individual values. Since we collected 20 pairs of individual values, always two pairs for one difference, we take the first value of a pair to be inserted in the system of equations. Wrong allocations will have contradictions somewhere in the equations or at least in the keystream for clock 7 and are dispelled this way. This would result in a time complexity of 2^{10} clocks. For the right allocation at clock 6 we can clock backwards and receive the secret key with a time complexity of six clocks.

At the beginning we guessed $\Delta R1^{-1}$ with 2^{26} possibilities and Δb_{10}^{-2} with two possibilities. Since we have the constraint on $\Delta R1^{-1}$ with msb=smsb=0, we have to execute our attack 2^2 times (for example with different Δd) that this case happens. The overall time complexity in K2$^{\oplus}$ clocks is

$$2^2 \cdot 2^{26} \cdot 2 \cdot (28.5 + 2^{10} + 6) \approx 2^{39.05} \quad .$$

The needed keystream amounts to two words per clock for seven clocks for each pair (K, IV_a) and (K, IV_b) which yields 28 keystream words. The memory requirements are roughly 30 words.

If we would restrict the difference Δd to a word with three bytes equal to zero and one non-zero byte, we would get a time complexity of

$$2^2 \cdot 2^5 \cdot 2 \cdot (28.5 + 2^{10} + 6) \approx 2^{18.05}$$

K2$^{\oplus}$ clocks. The keystream and memory requirements remain.

6.2.4 Reduced Initialization of 7 Clocks

Now we want to extend the attack to seven initialization clocks. The big problem is the feedback from the NLF to $FSR-A$ and the DFC. Table 6.3 shows the propagation of difference Δd and all unknown differences as question marks. Before we can start with any computation we have to guess some differences to get rid of some question marks in this table.

clock	FSR-B											FSR-A					NLF			
ini	10	9	8	7	6	5	4	3	2	1	0	4	3	2	1	0	L1	L2	R1	R2
								d												
1									d											
2										d										
3	d										d									
4	?	d										d								
5	?	?	d									d	d						?	
6	?	?	?	d								?	d	d					?	?
7	?	?	?	?	d							?	?	d	d		?		?	?

Table 6.3: Propagation of the difference Δd through K2$^{\oplus}$ during the seven initialization clocks

For $FSR-B$ we need to guess Δb_{10}^{-3} of initialization clock 4. In its update equation all differences are zero except for $\Delta b_0^{-4} = \Delta d$, for which we need to guess the choice of the multiplier which depends on the smsb of a_2^{-4}. Thus, we have two possibilities for Δb_{10}^{-3}. We denote the guessed Δb_{10}^{-3} with Δk_0. From the update equations we derive $\Delta b_{10}^{-2} = \Delta b_{10}^{-3} = \Delta k_0$. Similarly, we have to guess Δb_{10}^{-1} with two possibilities and denote it with Δk_1.

For the NLF we need to guess $\Delta R1^{-2}$ because it is fed back into $FSR-A$. We have $\Delta R1^{-2} \overset{\text{out}}{=\!\!\Delta} Sub(\Delta L2^{-3} \oplus \Delta b_9^{-3}) \overset{\text{out}}{=\!\!\Delta} Sub(0 \oplus \Delta d)$. Thus, we know the input difference and have 2^{28} possibilities for the output difference $\Delta R1^{-2}$. We can only employ those values of $\Delta R1^{-2}$ which have msb and smsb equal to zero because otherwise we will have differences in the bits for the DFC. This reduces the possibilities for $\Delta R1^{-2}$ to 2^{26} but we have to try the whole attack 2^2 times to let the case msb=smsb=0 happen. We denote the guessed $\Delta R1^{-2}$ with Δk_2. This fixes the input-output difference $\Delta Lb^{-3} \rightarrow Sub \rightarrow \Delta R1^{-2} = \Delta k_2$ yielding 16.51 pairs of individual values and following 8.255 possibilities for $\Delta R2^{-1}$. Furthermore, we guess $\Delta R1^{-1}$ with 2^{28} possibilities due to the known input difference in $\Delta R1^{-1} \overset{\text{out}}{=\!\!\Delta} Sub(\Delta L2^{-2} \oplus \Delta b_9^{-2}) \overset{\text{out}}{=\!\!\Delta} Sub(0 \oplus \Delta k_0)$. The sum $\Delta R1^{-1} \oplus \Delta R2^{-1}$ must have a zero in the msb and smsb because we do not want to have a difference in the bits for the DFC. This reduces the possibilities for $\Delta R1^{-1}$ to 2^{26} but we have to execute our attack 2^2 more times. We denote the guessed $\Delta R1^{-1}$ with Δk_3. This determines the input-output difference $\Delta Lb^{-2} \rightarrow Sub \rightarrow \Delta R1^{-1} = \Delta k_3$ yielding 16.51 pairs of individual values and 8.255 possibilities for $\Delta R2^0$.

We update Table 6.3 with all guessed differences and get Table 6.4.

Now we have a look at the keystream equations for clock 0. The difference Δb_{10}^0 has two possibilities because in its update equation we need to guess the multiplier for Δb_8^{-1} depending on the msb of a_2^{-1}. We can rewrite the

clock	FSR-B											FSR-A					NLF			
ini	10	9	8	7	6	5	4	3	2	1	0	4	3	2	1	0	L1	L2	R1	R2
								d												
1									d											
2										d										
3	d										d									
4	k_0	d										d								
5	k_0	k_0	d									d	d						k_2	
6	k_1	k_0	k_0	d								k_2	d	d					k_3	?
7	?	k_1	k_0	k_0	d							?	k_2	d	d		?		?	?

Table 6.4: Propagation of the known or guessed differences through K2$^{\oplus}$ during the seven initialization clocks

equation for Δz_0^H in the following way:

$$\begin{aligned} & & \Delta z_0^H &= \Delta b_{10}^0 \oplus \Delta L1^0 \oplus \Delta L2^0 \oplus \Delta a_0^0 \\ \Leftrightarrow & & \Delta L1^0 &= \Delta b_{10}^0 \oplus \Delta L2^0 \oplus \Delta a_0^0 \oplus \Delta z_0^H \\ \Leftrightarrow & \overset{\text{out}}{\Delta}\, Sub(\Delta R2^{-1} \oplus \Delta b_4^{-1}) &= \Delta b_{10}^0 \oplus 0 \oplus 0 \oplus \Delta z_0^H \\ \Leftrightarrow & \overset{\text{out}}{\Delta}\, Sub(\Delta R2^{-1} \oplus 0) &= \Delta b_{10}^0 \oplus \Delta z_0^H \ . \end{aligned}$$

We insert the 8.255 possibilities for $\Delta R2^{-1}$ and the two possibilities for Δb_{10}^0. Afterwards, we undo the MixColumns operation and check bytewise whether the value at the right side is a valid one for the left side. The time complexity is about four clocks and only one triple $(\Delta L1^0, \Delta R2^{-1}, \Delta b_{10}^0)$ will remain due to $\frac{2^{28} \cdot 2 \cdot 8.255}{2^{32}} \approx 1$. The $\Delta L1^0$ fixes the input-output difference $\Delta Rb^{-1} \rightarrow Sub \rightarrow \Delta L1^0$ which results in 8.255 possibilities for $\Delta L2^1$. The $\Delta R2^{-1}$ determines the two pairs of individual values for the sequence $\Delta Lb^{-3} \rightarrow Sub \rightarrow \Delta R1^{-2} \rightarrow Sub \rightarrow \Delta R2^{-1}$. Also, $\Delta R2^{-1}$ fixes $\Delta a_4^0 = \Delta a_3 {-} 1 \oplus \Delta R2^{-1} \oplus \Delta R1^{-1} \oplus \Delta a_4^{-1}$. For Δz_0^L we know the input difference for $\Delta R1^0 = \overset{\text{out}}{\Delta}\, Sub(\Delta L2^{-1} \oplus \Delta b_9^{-1}) = \overset{\text{out}}{\Delta}\, Sub(0 \oplus \Delta k_0)$ as well as the 8.255 possibilities for $\Delta R2^0$ and Δa_4^0. Therefore, we can do exactly the same as described for Δz_0^L at clock 0 in the attack on six initialization clocks on page 82. From now on all differences in $FSR{-}A$ are known and propagate linearly because there is no more feedback from the NLF to $FSR{-}A$.

For the clocks from 1 to 4 we perform the same computations as for clock 2 in the attack on six initialization clocks described on page 82. For clock 5 we discard Δz_5^H because we might have msb$\neq$0 and / or smsb$\neq$0 for Δa_2^4 which results in differences in the DFC. For Δz_5^L we do the same as in the previous clocks.

Now we have collected enough individual values for the NLF. So far, the time complexity for the computations is $4 + 4 \cdot (4 + 2) + 2 = 30$ clocks.

For each key-IV pair we have a system of keystream and update equations. We select one of them for the insertion of these individual values. Since we collected 20 pairs of individual values, always two pairs for one difference, we take the first value of a pair to be inserted in the system of equations. Wrong allocations will have contradictions somewhere in the equations or at least in the keystream for clock 6 and are dispelled this way. This would result in a time complexity of 2^{10} clocks. For the right allocation at clock 0 we can clock backwards and receive the secret key with time complexity of seven clocks.

At the beginning we guessed Δb_{10}^{-3} and Δb_{10}^{-2} each with two possibilities. Furthermore, we guessed $\Delta R1^{-2}$ and $\Delta R1^{-1}$ each with 2^{26} possibilities. Since we have the constraint on $\Delta R1^{-2}$ with msb=smsb=0 and the constraint on $\Delta R1^{-1}$, we have to execute our attack $2^2 \cdot 2^2$ times (for example with different Δd) that these cases happen. The overall time complexity in K2$^{\oplus}$ clocks is

$$2 \cdot 2 \cdot 2^{26} \cdot 2^{26} \cdot 2^2 \cdot 2^2 \cdot (30 + 2^{10} + 7) \approx 2^{68.05} \ .$$

The needed keystream amounts to two words per clock for six clocks for each pair (K, IV_a) and (K, IV_b) which yields 24 keystream words. The memory requirements are roughly 30 words.

If we would restrict the difference Δd to a word with three bytes equal to zero and one non-zero byte, we would get a time complexity of

$$2 \cdot 2 \cdot 2^5 \cdot 2^{26} \cdot 2^2 \cdot 2^2 \cdot (30 + 2^{10} + 7) \approx 2^{47.05}$$

K2$^{\oplus}$ clocks. The keystream and memory requirements remain.

The summary of our results is given in Table 6.5 with time in K2$^{\oplus}$ clocks and keystream and memory in words. We were not able to extend this attack to eight initialization clocks because we have not found a way to handle the unknown differences from the NLF fed back into $FSR-A$ and $FSR-B$. Furthermore, non-zero differences in the bits of the DFC will occur which makes the computation of the differences of $FSR-B$ impossible.

differential chosen IV attack on K2$^{\oplus}$	keystream	time	memory
5 initialization clocks	28	$2^{10.05}$	30
6 initialization clocks	28	$2^{39.05}$	30
with restriction of Δd to 1 byte	28	$2^{18.05}$	30
7 initialization clocks	24	$2^{68.05}$	30
with restriction of Δd to 1 byte	24	$2^{47.05}$	30

Table 6.5: Summary of our results on K2$^{\oplus}$

6.3 Chosen IV Distinguishing Attack

We now consider K2$^{\oplus}$ with the number of initialization clocks reduced to seven. To distinguish the cipher from a random function, we build a multiset over all possible 2^{32} values of one word using 2^{32} different key-IV pairs and check whether the XORed sum over all first keystream words z_0^H is equal to zero which holds with probability one. For a random function the probability is 2^{-32} that this sum is zero.

For all 2^{32} key-IV pairs we take the same unknown key K. The four words of the $IV = [IV_0, IV_1, IV_2, IV_3]$ are loaded in the $FSR{-}B$, where the word IV_1 loaded in b_3 takes the longest time until it enters the NLF. For this word IV_1 we make a multiset in a way that all values $[0, 2^{32} - 1]$ occur exactly once. We emphasize that we need the multiset in ascending order starting with zero. Then we know that all IV_1 values in the first half have msb equal to zero whereas all IV_1 values in the second half have msb equal to one. We will use this fact about the msb later. The multiset in IV_1 yields 2^{32} different IVs $IV^i = [IV_0, i, IV_2, IV_3],\ i = 0, \ldots, 2^{32} - 1$ with arbitrary words $IV_{(0,2,3)}$. For each pair (K, IV^i) we run the K2$^{\oplus}$ cipher with seven initialization clocks and get the first keystream word ${}^i z_0^H$.

Now we explain why the XORed sum over all keystream words is equal to zero. Our goal is to prove the multiset propagation through K2$^{\oplus}$ as shown in Table 6.6.

clock	FSR-B											FSR-A					NLF			
ini	10	9	8	7	6	5	4	3	2	1	0	4	3	2	1	0	L1	L2	R1	R2
0								M^s												
1									M^s											
2										M^s										
3	M^s										M^s									
4	S0	M^s										M^s								
5	S0	S0	M^s									M^s	M^s						M^1	
6	S0	S0	S0	M^s								S0	M^s	M^s					?	M^2
7	S0	S0	S0	S0	M^s							?	S0	M^s	M^s		M^3		?	?

Table 6.6: Propagation of the multiset through K2$^{\oplus}$ during the seven initialization clocks

In this table we only show the multiset and its behavior. All values which are the same in all 2^{32} key-IV pairs are omitted (empty in the table). We put the '?' for those sets we do not know anything about. With the symbol 'M^s' we denote the multiset in the starting order. The symbols 'M^1', 'M^2' and 'M^3' denote different multisets. In each of them each value $[0, 2^{32} - 1]$ occurs exactly once but we do not know in which order. Thus, also the XORed sums

over these multisets are zero. The symbol 'S0' denotes a multiset, where the characteristic that each value $[0, 2^{32}-1]$ occurs exactly once is lost, but the feature that it sums up (XORed) to zero still remains. To prove this feature for the 'S0' multisets it is sufficient to consider the sets in b_{10}^t and a_4^t. The update equation for the XORed sum over all ${}^i b_{10}^t$ is

$$\sum_{i=0}^{2^{32}-1} {}^i b_{10}^t = \sum_{i=0}^{2^{32}-1} \Big({}^i m_{smsb}^{t-1}\, {}^i b_0^{t-1} \oplus {}^i b_1^{t-1} \oplus {}^i b_6^{t-1} \oplus {}^i m_{msb}^{t-1}\, {}^i b_8^{t-1} \oplus {}^i b_{10}^{t-1} \oplus {}^i L2^{t-1} \oplus {}^i L1^{t-1} \oplus {}^i a_0^{t-1} \Big) \ . \quad (6.1)$$

Here, the variable ${}^i m_{msb}^{t-1}$ denotes the multiplier depending on the msb of ${}^i a_2^{t-1}$. In particular, ${}^i m_{msb}^{t-1} = \alpha_3$ if the msb of ${}^i a_2^{t-1}$ is equal to one and ${}^i m_{msb}^{t-1} = 1$ otherwise. Likewise, the variable ${}^i m_{smsb}^{t-1}$ denotes the multiplier depending on the smsb of ${}^i a_2^{t-1}$. In particular, ${}^i m_{smsb}^{t-1} = \alpha_1$ if the smsb of ${}^i a_2^{t-1}$ is equal to one and ${}^i m_{smsb}^{t-1} = \alpha_2$ otherwise. The update equation for the XORed sum over all ${}^i a_4^t$ is

$$\sum_{i=0}^{2^{32}-1} {}^i a_4^t = \sum_{i=0}^{2^{32}-1} \Big(\alpha_0\, {}^i a_0^{t-1} \oplus {}^i a_3^{t-1} \oplus {}^i b_0^{t-1} \oplus {}^i R2^{t-1} \oplus {}^i R1^{t-1} \oplus {}^i a_4^{t-1} \Big) \ . \quad (6.2)$$

From now on, we always mean XORed sum when we write sum or the sigma sign.

For clock 1 and 2 our starting multiset is only shifted in the $FSR-B$. We do not mention the shifts of a multiset in the next clocks anymore. At clock 3 we compute the sum over all ${}^i b_{10}^3$. We see from Table 6.6 that the choice for the multipliers is constant due to the constant value of a_2^2. Thus, we do not know which multiplier is chosen but we know it is the same for all 2^{32} key-IV pairs. Table 6.6 also shows that nearly all summands are constants, too, which means their sums are zero. We omit all zero sums reducing (6.1) to

$$\sum_{i=0}^{2^{32}-1} {}^i b_{10}^3 = \sum_{i=0}^{2^{32}-1} {}^i b_1^2 = \sum_{i=0}^{2^{32}-1} {}^i b_3^0 = \sum_{i=0}^{2^{32}-1} i = 0 \ , \quad (6.3)$$

because it is the sum of our starting multiset and therefore zero. For clock 4 the choice for the multipliers in (6.1) is constant as well. We leave all zero

sums out (empty values in Table 6.6) and yield

$$\begin{aligned}\sum_{i=0}^{2^{32}-1} {}^{i}b_{10}^{4} &= m_{smsb}^{3} \sum_{i=0}^{2^{32}-1} {}^{i}b_{0}^{3} \oplus \sum_{i=0}^{2^{32}-1} {}^{i}b_{10}^{3} \\ &= m_{smsb}^{3} \sum_{i=0}^{2^{32}-1} {}^{i}b_{3}^{0} \oplus \sum_{i=0}^{2^{32}-1} {}^{i}b_{3}^{0} \\ &= m_{smsb}^{3} \sum_{i=0}^{2^{32}-1} i \oplus \sum_{i=0}^{2^{32}-1} i = 0 \end{aligned} \tag{6.4}$$

due to (6.3). At this clock our starting multiset enters the $FSR-A$ because after omitting all zero sums (6.2) gives

$$\sum_{i=0}^{2^{32}-1} {}^{i}a_{4}^{4} = \sum_{i=0}^{2^{32}-1} {}^{i}b_{0}^{3} = \sum_{i=0}^{2^{32}-1} {}^{i}b_{3}^{0} = \sum_{i=0}^{2^{32}-1} i = 0 \ . \tag{6.5}$$

Looking at clock 5 the choice for the multipliers is constant due to the constant value of a_2^4. Table 6.6 shows that nearly all sums are zero which reduces (6.1) with (6.4) to

$$\sum_{i=0}^{2^{32}-1} {}^{i}b_{10}^{5} = \sum_{i=0}^{2^{32}-1} {}^{i}b_{10}^{4} = 0 \ . \tag{6.6}$$

For $FSR-A$ we get from (6.2) and (6.5)

$$\sum_{i=0}^{2^{32}-1} {}^{i}a_{4}^{5} = \sum_{i=0}^{2^{32}-1} {}^{i}a_{4}^{4} = 0 \ . \tag{6.7}$$

At this clock our starting multiset also enters the NLF in word $R1$ yielding with (6.3)

$$\begin{aligned}\sum_{i=0}^{2^{32}-1} {}^{i}R1^{5} &= \sum_{i=0}^{2^{32}-1} Sub\left({}^{i}L2^{4} \oplus {}^{i}b_{9}^{4}\right) \\ &= \sum_{i=0}^{2^{32}-1} Sub\left({}^{i}L2^{4} \oplus {}^{i}b_{10}^{3}\right) \\ &= \sum_{i=0}^{2^{32}-1} Sub\left({}^{i}L2^{4} \oplus i\right) = 0 \end{aligned} \tag{6.8}$$

due to the constant value of $L2^4$. Thus, the order of our starting multiset is destroyed but the feature that each value $[0, 2^{32}-1]$ occurs exactly once remains which results in a zero sum.

For clock 6 the choice for the multipliers in (6.1) is constant as well. Thus, we can lower (6.1) with (6.3) and (6.6) to

$$\begin{aligned}\sum_{i=0}^{2^{32}-1} {}^{i}b_{10}^{6} &= m_{msb}^{5} \sum_{i=0}^{2^{32}-1} {}^{i}b_{8}^{5} \oplus \sum_{i=0}^{2^{32}-1} {}^{i}b_{10}^{5} \\ &= m_{msb}^{5} \sum_{i=0}^{2^{32}-1} {}^{i}b_{10}^{3} \oplus \sum_{i=0}^{2^{32}-1} {}^{i}b_{10}^{5} \\ &= m_{msb}^{5} \sum_{i=0}^{2^{32}-1} i \oplus \sum_{i=0}^{2^{32}-1} {}^{i}b_{10}^{5} = 0 \ . \end{aligned} \tag{6.9}$$

For $FSR-A$ we can reduce (6.2) with (6.5), (6.8) and (6.7) to

$$\begin{aligned}\sum_{i=0}^{2^{32}-1} {}^{i}a_{4}^{6} &= \sum_{i=0}^{2^{32}-1} {}^{i}a_{3}^{5} \oplus \sum_{i=0}^{2^{32}-1} {}^{i}R1^{5} \oplus \sum_{i=0}^{2^{32}-1} {}^{i}a_{4}^{5} \\ &= \sum_{i=0}^{2^{32}-1} {}^{i}a_{4}^{4} \oplus \sum_{i=0}^{2^{32}-1} {}^{i}R1^{5} \oplus \sum_{i=0}^{2^{32}-1} {}^{i}a_{4}^{5} = 0 \ . \end{aligned} \tag{6.10}$$

At this clock we have the first unknown set in the NLF because with (6.6) and (6.4) we get

$$\begin{aligned}\sum_{i=0}^{2^{32}-1} {}^{i}R1^{6} &= \sum_{i=0}^{2^{32}-1} Sub\left({}^{i}L2^{6} \oplus {}^{i}b_{9}^{6}\right) \\ &= \sum_{i=0}^{2^{32}-1} Sub\left({}^{i}L2^{6} \oplus {}^{i}b_{10}^{5}\right) \\ &= \sum_{i=0}^{2^{32}-1} Sub\left({}^{i}L2^{6} \oplus m_{smsb}^{3} i \oplus i\right) \ , \end{aligned} \tag{6.11}$$

where we do not know how the summands in the Sub function behave. Therefore, we have no clue what outcome the entire sum might have. The word $R2^6 = Sub(R1^5)$ is a multiset, because $R1^5$ is a multiset, in which each value $[0, 2^{32}-1]$ occurs exactly once, and the Sub function preserves this property. Thus, we do not know the order of $R2^6$ but we know that the sum is zero. At clock 7 we see that we have only three zero sums in Table 6.6. This reduces (6.1) to

$$\sum_{i=0}^{2^{32}-1} {}^{i}b_{10}^{7} = \sum_{i=0}^{2^{32}-1} {}^{i}m_{smsb}^{6}\, {}^{i}b_{0}^{6} \oplus \sum_{i=0}^{2^{32}-1} {}^{i}m_{msb}^{6}\, {}^{i}b_{8}^{6} \oplus \sum_{i=0}^{2^{32}-1} {}^{i}b_{10}^{6} \ . \tag{6.12}$$

The first two summands depend on the value of a_2^6. From the update equation for $a_2^6 = a_4^4$ we know

$$\sum_{i=0}^{2^{32}-1} {}^i a_4^4 \;=\; \sum_{i=0}^{2^{32}-1} \Big(\alpha_0 {}^i a_0^3 \;\oplus\; {}^i a_3^3 \;\oplus\; {}^i b_0^3 \;\oplus\; {}^i R2^3 \;\oplus\; {}^i R1^3 \;\oplus\; {}^i a_4^3 \Big) \;.$$

The values for $a_3^3,\ a_0^3,\ R2^3,\ R1^3$ and a_4^3 remain constant for all pairs. We denote the sum of them with $C = \alpha_0 a_0^3 \oplus a_3^3 \oplus R2^3 \oplus R1^3 \oplus a_4^3$ which does not depend on i and is unknown to us. With this simplification and (6.5), we obtain

$$\sum_{i=0}^{2^{32}-1} {}^i a_4^4 \;=\; \sum_{i=0}^{2^{32}-1} \Big(i \oplus C \Big) \;. \tag{6.13}$$

As a result of the multiset we know that in each bit of i, the number of ones and zeros occurring is exactly 2^{31} which is an even number. This means that the smsb of a_4^4 has 2^{31} ones and 2^{31} zeros. Taking this and the fact that the value of b_0^6 remains constant we obtain for the first summand of (6.12)

$$\sum_{i=0}^{2^{32}-1} {}^i m_{smsb}^6 {}^i b_0^6 \;=\; \sum_{i=0}^{2^{31}-1} \alpha_1 b_0^6 \;\oplus\; \sum_{i=0}^{2^{31}-1} \alpha_2 b_0^6 = 0 \;. \tag{6.14}$$

Altogether, (6.12) simplifies with (6.9) and (6.14) to

$$\sum_{i=0}^{2^{32}-1} {}^i b_{10}^7 \;=\; \sum_{i=0}^{2^{32}-1} {}^i m_{msb}^6 {}^i b_8^6 \;=\; \sum_{i=0}^{2^{32}-1} {}^i m_{msb}^6 {}^i b_{10}^4 \;, \tag{6.15}$$

where the choice of the multiplier depends on the msb of the value $a_2^6 = a_4^4$ and the value of b_{10}^4 does not remain constant. At the beginning we emphasized the ascending order of our multiset. This means that the first half of our multiset has msb zero and the second half has msb one. Accordingly, the msb of all ${}^i a_4^4$ with $i = 0, \ldots, 2^{31} - 1$ is the msb of the unknown constant C shown in (6.13), whereas the msb of all ${}^i a_4^4$ with $i = 2^{31}, \ldots, 2^{32} - 1$ is the opposite of the msb of the unknown constant C. Thus, we divided the sum into a first and a second half. We need to check whether we can also divide the set of ${}^i b_{10}^4$. From (6.4) we know

$$\sum_{i=0}^{2^{32}-1} {}^i b_{10}^4 \;=\; m_{smsb}^3 \sum_{i=0}^{2^{32}-1} i \;\oplus\; \sum_{i=0}^{2^{32}-1} i \;.$$

The choice of the multipliers is constant but unknown to us. We can divide both sums into a first and a second half. Together with the two possibilities of the msb of a_4^4 (first half zero and second half one, or vice versa) we can

compute two values for (6.15)

$$X_1 = \left(m^3_{smsb} \sum_{i=0}^{2^{31}-1} i \oplus \sum_{i=0}^{2^{31}-1} i \right) \oplus \alpha_3 \left(m^3_{smsb} \sum_{i=2^{31}}^{2^{32}-1} i \oplus \sum_{i=2^{31}}^{2^{32}-1} i \right)$$

$$X_2 = \alpha_3 \left(m^3_{smsb} \sum_{i=0}^{2^{31}-1} i \oplus \sum_{i=0}^{2^{31}-1} i \right) \oplus \left(m^3_{smsb} \sum_{i=2^{31}}^{2^{32}-1} i \oplus \sum_{i=2^{31}}^{2^{32}-1} i \right) .$$

For the sums over exactly one ordered half of the multiset the property of summing up to zero is preserved. Accordingly, the values X_1 and X_2 are both zero which results in

$$\sum_{i=0}^{2^{32}-1} {}^i b^7_{10} = X_1 = 0 \quad \text{or} \quad \sum_{i=0}^{2^{32}-1} {}^i b^7_{10} = X_2 = 0 . \tag{6.16}$$

The unknown sum in $R1^6$ gives us an unknown sum in $R2^7 = Sub(R1^6)$ and an unknown sum in a^7_4 due to (6.2). In $R1^7$ we get an unknown sum for a similar reason as for $R1^6$ in (6.11). We also get a multiset in $L1^7$ due to

$$\sum_{i=0}^{2^{32}-1} {}^i L1^7 = \sum_{i=0}^{2^{32}-1} Sub\left({}^i R2^6 \oplus {}^i b^6_4 \right) = 0 , \tag{6.17}$$

where ${}^i b^6_4$ remains constant for all i. Thus, we have a multiset in $L1^7$ in which each value $[0, 2^{32} - 1]$ occurs exactly once, because the Sub function preserves this feature of the multiset in $R2^6$. Consequently, we have proven the assumed multiset propagation through K2$^{\oplus}$ in Table 6.6.

The keystream is produced at clock zero after the initialization. For the internal states we write the number of the initialization clock as superscript. We know about the sum of all words ${}^i z^H_0$

$$\sum_{i=0}^{2^{32}-1} {}^i z^H_0 = \sum_{i=0}^{2^{32}-1} \left({}^i b^7_{10} \oplus {}^i L2^7 \oplus {}^i L1^7 \oplus {}^i a^7_0 \right) .$$

For all 2^{32} key-IV pairs the values for $L2^7$ and a^7_0 remain constant for all pairs meaning the sum over them is zero. The values ${}^i L1^7$ form a multiset which sums up to zero and (6.16) shows the zero sum for ${}^i b^7_{10}$. Thus, we have

$$\sum_{i=0}^{2^{32}-1} {}^i z^H_0 = 0 . \tag{6.18}$$

We have shown that we can distinguish the K2$^{\oplus}$ from a random function with probability $1 - 2^{-32}$. We need the first keystream word for all 2^{32} pairs

(key, IV^i) with ($i = 0, \ldots, 2^{32}-1$). The time complexity is 2^{32} times an XOR of a word which is roughly $2^{27.6}$ clocks of K2$^{\oplus}$. The memory requirements are negligible.

We have not found a way to extend this attack to eight initialization clocks because in $L1^8$ we will get a set we do not know anything about coming from $R2^7$ and therefore resulting in a random sum.

6.4 Summary

We have shown a differential chosen IV attack with key-recovery on K2$^{\oplus}$ with five, six and seven initialization clocks. The time complexities for the three attacks are $2^{10.05}$, $2^{18.05}$ and $2^{47.05}$ clocks of K2$^{\oplus}$. All three attacks have the same memory requirements of 30 words. The attacks on five and six clocks need 28 words of keystream, and for seven clocks 24 keystream words are needed.

A chosen IV distinguishing attack on K2$^{\oplus}$ with seven initialization clocks is also presented. With a multiset and its predictable propagation through these seven clocks we can distinguish K2$^{\oplus}$ from a random function with a probability of $1-2^{-32}$. The complexity for this attack is $2^{27.6}$ clocks of K2$^{\oplus}$, negligible memory requirements and needed keystream of 2^{32} words.

We can only attack reduced versions of K2$^{\oplus}$ with seven initialization clocks out of 24. Thus, the security margin is large enough and these attacks do not pose a threat to the K2 stream cipher.

Chapter 7

Cryptanalysis of the Shrinking Generator with Linear Initialization

The shrinking generator [CKM93] is a classic stream cipher construction that is specified without any key loading or resynchronization mechanisms; the initial state is considered to be the secret key. This is probably one of the reasons why even though these schemes have been very intensively studied in the academic literature they are not widely used in practice.

In this chapter we show that a linear initialization with known feedback polynomials results in a very weak cipher. This coincides with similar findings for filter generators and combiners with memory [ALP04] and with chosen IV attacks on irregularly clocked ciphers A5/1 [EJ03, BB05] and DECT [NTW10] amongst others.

We assume that the attacker has access to output streams of multiple states that are differentially related, i.e. not only does the attacker know the output sequence (z_i^0) for a secret initial state σ_0 but also output sequences (z_i^j) for states $\sigma_j = \sigma_0 \oplus \Delta_j$ together with the corresponding differences Δ_j. This can occur if the internal state of the stream cipher is not only initialized with a key but also with an initial value (IV) that is transmitted in clear, an approach that is commonly used when the communication secured by the stream cipher is packetized. Then the attacker can use the known differences to determine the internal state and thus the secret key.

The chapter is organized as follows: In Section 7.1 we briefly describe the shrinking generator. In Section 7.2 we show the known IV key-recovery attack on the shrinking generator with linear initialization and known feedback polynomials. The summary is given in Section 7.3.

7.1 Description of the Shrinking Generator

The shrinking generator is a stream cipher construction proposed by Coppersmith, Krawczyk and Mansour [CKM93] in 1993. Two LFSRs A and S are used to generate a so-called "shrunken sequence" by conditionally emitting the output bit of A only if the output bit of the selector S in a given clock equals 1; if the output bit of S equals 0, no output is produced. The output sequence of the shrinking generator will be denoted by (z_i). We denote by (s_i) the output sequence and by n_S the length of LFSR S. Similarly, (a_i) denotes the output sequence and n_A the length of LFSR A. For optimal security, $gcd(n_S, n_A) = 1$ should hold; in this case the period of the generator is $(2^{n_A} - 1) \cdot 2^{n_S - 1}$.

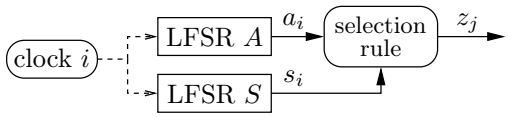

Figure 7.1: Shrinking Generator

In the original description the initial state and sometimes the feedback polynomials as well are assumed to be the secret key. The use of an IV is not mentioned nor is a method for loading and / or initialization given.

We consider the shrinking generator with a linear key-IV setup and known feedback polynomials. The shrinking generator is initialized with a key K and an IV V by sequentially clocking the bits of K and V into both LFSRs. We have no restriction whether the key or the IV is clocked into the LFSRs first. After the key and IV loading a number of blank rounds can be executed, i.e. the shrinking generator is clocked without producing output.

7.2 Known IV Key-Recovery Attack

In this section we demonstrate a known IV key-recovery attack against the shrinking generator with a linear key-IV setup and known feedback polynomials. The attacker has access to v different keystreams $(z_i)^j$ with $j = 1, \ldots, v$ which are produced by $v \geq 2$ known IVs and always the same secret key. Of course, the corresponding sequences $(s_i)^j$ and $(a_i)^j$ produced by the LFSRs S and A are unknown to the attacker.

We can compute all differences between the known IVs and denote their amount with $w = \sum_{i=1}^{v-1} i$. With each IV difference we start the shrinking generator to produce the difference sequences for the LFSRs S denoted with $\Delta(s_i)^h$ and the difference sequences for the LFSRs A denoted with $\Delta(a_i)^h$ where $h = 1, \ldots, w$.

We can determine the first n_S output bits of the unknown sequences $(s_i)^j$ sequentially bit by bit using the knowledge of $\Delta(s_i)^h$, $\Delta(a_i)^h$ and the differences in the given keystreams. This allows us to recover the initial states of the corresponding LFSRs S. Subsequently, we can recover the initial state of each LFSR A using only the corresponding LFSR S and the keystream. With the recovered LFSR S we can assign the clock to each keystream bit when it was produced by the LFSR A. This gives us a system of linear equations in the unknown bits of the initial state of LFSR A. We need n_A bits of the keystream to yield enough equations. This corresponds to $2\,n_A$ clocks because roughly half of the output bits of LFSR A are discarded. We solve the system in $\mathcal{O}(n_A^3)$ time and $\mathcal{O}(n_A^2)$ memory complexity and yield the initial state of the LFSR A. Now we know the initial state of the LFSRs S and A and therefore the secret key.

To determine the output bits of the unknown sequences $(s_i)^j$ we consider each clock separately. At each clock t we compute all differences between the first bits of the sequences $(z_i)^j$ and denote them with Δz_1^h with $h = 1, \ldots, w$. Now we look at the Δs_t^h and can distinguish three cases where each case has different probabilities.

The first case occurs when all $\Delta s_t^h = 0$ which means all unknown s_t^j have the same value, either zero or one. For all $h = 1, \ldots, w$ we check whether $\Delta a_t^h = \Delta z_1^h$. If at least one h exists with $\Delta a_t^h \neq \Delta z_1^h$ then we conclude that all unknown s_t^j must be zero because the differences in the output of the LFSRs A do not match the differences in the keystream. If all differences of the output and the keystream match we can not decide whether this happens by coincidence or because all $s_t = 1$.

The second case occurs when $v - 1$ out of w of the Δs_t^h equal one which means that one of the s_t^j has the opposite value than the other s_t^j. We consider only these h for which $\Delta s_t^h = 0$ and check for these h whether $\Delta a_t^h = \Delta z_1^h$. If among these h at least one exists with $\Delta a_t^h \neq \Delta z_1^h$ then we conclude that the corresponding s_t^j must be zero because the differences in the output of the LFSRs A do not match the differences in the keystream. If all these differences match we can not decide whether this happens coincidentally or because the corresponding $s_t = 1$.

The third case occurs when the Δs_t^h can be divided into three subsets. The first subset contains $\Delta s_t^h = 0$ so that the corresponding s_t^j all have the same value. The second subset contains $\Delta s_t^h = 0$ as well so that the corresponding s_t^j all have the same value but this value is opposite to the value of the s_t^j belonging to the first subset. Consequentially, the third subset contains all $\Delta s_t^h = 1$. For the first and second subset we check the corresponding $\Delta a_t^h = \Delta z_1^h$. If for one subset all differences match and for the other subset at least one inequation occurs then we can determine that all s_t^j belonging to the subset with the inequation must be zero and the s_t^j belonging to the other subset are one. If all differences match we can

not decide whether the s_t^j belonging to the first subset are zero and the s_t^j belonging to the second subset are one or the other way around. It can also happen that for both subsets at least one inequation occurs which means the s_t^j belonging to the first subset must be zero as well as the s_t^j belonging to the second subset. This is a contradiction because the Δs_t^h constitute that not all s_t^j can have the same value. This contradiction can only appear if at a previous clock u we had to guess s_u^j equals zero or one. So we have the possibility to detect wrong guesses. If we had no guess in all previous clocks we will have a decision.

For two or three IVs we can not eliminate wrong guesses because the case for a contradiction never occurs. The amount of IVs is too small to yield a division into three subsets as needed for the third case.

At each clock t we can either decide which s_t^j is zero and which is one or we can not decide and therefore have to guess which s_t^j equals zero or one. If $s_t^j = 1$ the first bit of the corresponding keystream sequence $(z_i)^j$ is discarded and the next bit becomes the first one.

Table 7.1 shows the frequency of occurrence for the different cases for two up to nine IVs. The variable v denotes the number of IVs, p the amount of possibilities, g the guesses, d the decisions and c the contradictions. The probability to have a contradiction is given by r. The last column itemizes as subcategories the different cases with the division of the s_t^j into subsets. The value m denotes how often this division can occur and "$\alpha \,|\, \beta$" describes the exact division of the s_t^j into subsets. Please note that $\alpha \leq \beta$ and $\alpha + \beta = v$. For example for six IVs shows "$2 \,|\, 4$" the division where two of the s_t^j have the same value and the remaining four s_t^j have the same value but this one is opposite to the value of the previous two. The first subcategory contains the **first case** when all $\Delta s_t^h = 0$ thus all s_t^j have the same value which gives the division "$0 \,|\, v$". The second subcategory contains the **second case** when one of the s_t^j has the opposite value than the other s_t^j yielding the division "$1 \,|\, v - 1$". The remaining subcategories contain the different possibilities for the **third case** where each subset given by the division contains at least two s_t^j.

The more IVs are provided the better the attack works due to the increased probability for determining s_t^j and more cases to eliminate wrong guesses by a contradiction. For an arbitrary amount v of IVs the value p can be computed with $p = 2^{2(v-1)}$. We were not able to find a single equation to compute the sum d and c depending on the amount of IVs but a recursive description for the separate entries in Table 7.1. For d and c as well as g and m we define a new notation where the superscript gives the number of IVs and the bottom index denotes the division of the s_t^j into subsets. For example $d_{1|2}^3 = 2$ is the number of decisions for three IVs in the **second case** where one s_t^j has the opposite value than the other two.

IVs	Poss.	Sums		Prob.	Cases itemized										
v	p	$\sum g$	$\sum d$	$\sum c$	$r = \frac{\sum c}{p}$	$m\times\alpha\,\vert\,\beta$ g / d / c		$m\times\alpha\,\vert\,\beta$ g / d / c		$m\times\alpha\,\vert\,\beta$ g / d / c		$m\times\alpha\,\vert\,\beta$ g / d / c		$m\times\alpha\,\vert\,\beta$ g / d / c	…
2	2^2	3	1	0	0.0 %	1× 0 | 2 1 / 1 / 0		1× 1 | 1 2 / 0 / 0		—		—		—	…
3	2^4	7	9	0	0.0 %	1× 0 | 3 1 / 3 / 0		3× 1 | 2 2 / 2 / 0		—		—		—	…
4	2^6	15	43	6	9.4 %	1× 0 | 4 1 / 7 / 0		4× 1 | 3 2 / 6 / 0		3× 2 | 2 2 / 4 / 2		—		—	…
5	2^8	31	165	60	23.4 %	1× 0 | 5 1 / 15 / 0		5× 1 | 4 2 / 14 / 0		10× 2 | 3 2 / 8 / 6		—		—	…
6	2^{10}	63	571	390	38.1 %	1× 0 | 6 1 / 31 / 0		6× 1 | 5 2 / 30 / 0		15× 2 | 4 2 / 16 / 14		10× 3 | 3 2 / 12 / 18		—	…
7	2^{12}	127	1,869	2,100	51.3 %	1× 0 | 7 1 / 63 / 0		7× 1 | 6 2 / 62 / 0		21× 2 | 5 2 / 32 / 30		35× 3 | 4 2 / 20 / 42		—	…
8	2^{14}	255	5,923	10,206	62.3 %	1× 0 | 8 1 / 127 / 0		8× 1 | 7 2 / 126 / 0		28× 2 | 6 2 / 64 / 62		56× 3 | 5 2 / 36 / 90		35× 4 | 4 2 / 28 / 98	…
9	2^{16}	511	18,405	46,620	71.1 %	1× 0 | 9 1 / 255 / 0		9× 1 | 8 2 / 254 / 0		36× 2 | 7 2 / 128 / 126		84× 3 | 6 2 / 68 / 186		126× 4 | 5 2 / 44 / 210	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮		⋮		⋮		⋮		⋮	⋱

Table 7.1: Probability of contradictions

For the multiplier m we have the following recursive equations:

for the first subcategory	$m^v_{0\|v} = 1$
for the first entry of the second subcategory	$m^2_{1\|1} = 1$
for all entries with $\alpha \neq \beta$	$m^v_{\alpha\|\beta} = \frac{v}{\alpha} \cdot m^{v-1}_{\alpha-1\|\beta}$
for all entries with $\alpha = \beta$	$m^v_{\alpha\|\beta} = \frac{v}{2\alpha} \cdot m^{v-1}_{\alpha-1\|\beta}$

The number of guesses g can easily be extracted from the table:

for the first subcategory	$g^v_{0\|v} = 1$
for all other entries	$g^v_{\alpha\|\beta} = 2$
equation for the sum	$\sum g^v_{\alpha\|\beta} = 2^v - 1$

For the number of decisions d we have the following recursive equations:

for the first subcategory	$d^v_{0\|v} = 2^{v-1} - 1 = \sum g^{v-1}_{\alpha\|\beta}$
for the second subcategory	$d^v_{1\|v-1} = 2^{v-1} - 2 = 2 \cdot d^{v-1}_{0\|v-1}$
for all other entries	$d^v_{\alpha\|\beta} = 2^{\alpha-1} + d^{v-1}_{\alpha-1\|\beta} = 2^{v-\alpha-1} + d^{v-1}_{\alpha\|\beta-1}$

For the number of contradictions c the recursive equations are:

for the first and second subcategory	$c^v_{0\|v} = c^v_{1\|v-1} = 0$
for the third subcategory	$c^v_{2\|v-2} = 2^{v-2} - 2$
for all other entries	$c^v_{\alpha\|\beta} = \frac{2^{\beta-1}-1}{2^{\beta-2}-1} \cdot c^{v-1}_{\alpha-1\|\beta}$

The sum for d and c can be computed with these equations.

To mount the known IV attack with two or three IVs is useless because we would have to guess too many bits of the sequences $(s_i)^j$. Even with four IVs the probability of getting a guess with 23.4 % is much higher than the probability of getting a contradiction with 9.4 % which results in an exponential time complexity to determine the bits of the sequences $(s_i)^j$. For five IVs we have a probability of 23.4 % to get a contradiction which is nearly twice the probability of 12.1 % to get a guess. Therefore, we conclude that the time complexity to recover the bits of the sequences $(s_i)^j$ would be in $\mathcal{O}(n_S)$ if at least five IVs are provided.

We assume both LFSRs having nearly the same length. Then the overall time complexity for our attack is dominated by $\mathcal{O}(n_A^3)$ as well as the memory requirements by $\mathcal{O}(n_A^2)$. The needed keystream is roughly $2n_A$ bits for each IV.

Note on the Self-Shrinking Generator with Known Feedback Polynomials

The self-shrinking generator was presented by Meier and Staffelbach [MS94] in 1994. The generator uses only one LFSR L and is specified without any key loading or resynchronization mechanisms. The initial state is considered to be the secret key. The output bits of the LFSR are grouped to pairs (l_{i+1}, l_i) where the bit l_{i+1} decides whether the bit l_i is an output bit of the self-shrinking generator or not. If $l_{i+1} = 1$ the bit l_i is the output bit of the self-shrinking generator. If $l_{i+1} = 0$ the bit l_i is discarded.

We can determine the initial state of the self-shrinking generator in a similar way as for the shrinking generator. By dividing the sequence produced by the LFSR L into a "selector" sequence and a "non-shrunken" sequence, we are in the same place as for the shrinking generator. We can recover the bits of the "selector" sequence one by one. The keystream contains roughly half of the bits of the "non-shrunken" sequence and with the recovered "selector" bits we also know the clock for each bit. Altogether we can determine roughly $\frac{3}{4}$ of the bits in the sequence produced by L. Therefore we need to know the bits for $\frac{4}{3}n_L$ clocks on average to yield a system of n_L linear equations in the unknown initial state variables. The time complexity to solve this system is in $\mathcal{O}(n_L^3)$ and the memory requirements are in $\mathcal{O}(n_L^2)$. Compared to these complexities the needed time and memory for the determination of the "selector" bits are negligible. The needed keystream is roughly $\frac{1}{3}n_L$ bits for each IV.

7.3 Summary

In this chapter we described a known IV key-recovery attack on the shrinking generator with known linear feedback polynomials and a linear key-IV setup. Despite the irregular clocking we are able to break the cipher in $\mathcal{O}(n_A^3)$ time complexity and $\mathcal{O}(n_A^2)$ memory requirements using at least five IVs. The design of the self-shrinking generator is related to the design of the shrinking generator. Therefore, this cipher can also be broken with at least five IVs. The time complexities is in $\mathcal{O}(n_L^3)$ and the memory requirements are in $\mathcal{O}(n_L^2)$.

Chapter 8

Conclusion

Even the most successful attacks started
with pointing out a small weakness.
based loosely on Confucius

This thesis presents cryptanalysis of several stream ciphers, especially known or chosen IV key-recovery attacks against their initialization methods.

If the update of the internal state is the same during initialization and keystream generation or if a cipher does not have an initialization method then it is worth to consider a slide attack. We describe slid pairs for Salsa20 and Trivium. The special structure of their starting states limits the amount of slid pairs so that the probability of getting such a slid pair by chance is too small to pose a real threat on these ciphers.

For both SNOW ciphers the addition modulo 2^{32} is replaced by an XOR to retain only the FSM as a non-linear part. If this non-linear part has no influence on the linear part during the initialization the construction is vulnerable. When the FSM affects the LFSR we are able to attack roughly half of the initialization clocks.

The K2 cipher has a new design aspect to increase non-linearity. The feedback function of one FSR is dynamically chosen by the other FSR among four different linear feedback functions. This makes differential cryptanalysis even more difficult than only dealing with the non-linearity introduced by the FSM. We are only able to attack seven out of 24 initialization clocks for a version where all additions modulo 2^{32} are replaced by XORs.

A rather old design is the shrinking generator where the output bit of one LSFR decides whether the output bit of the second LFSR becomes the keystream. This results in an irregularly clocked keystream sequence. We consider known feedback polynomials and a linear initialization with an IV and are able to break the shrinking generator despite the irregular clocking using only a few known IVs.

Apart from the shrinking generator the analyzed stream ciphers are rather recent designs. We point out some weaknesses but are not able to break these ciphers. Time will tell whether they withstand cryptanalytic attacks or get broken.

Bibliography

[ACG+06] F. Armknecht, C. Carlet, P. Gaborit, S. Künzli, W. Meier, and O. Ruatta. *Efficient Computation of Algebraic Immunity for Algebraic and Fast Algebraic Attacks*. In S. Vaudenay (ed.), EUROCRYPT, volume 4004 of Lecture Notes in Computer Science, pages 147–164. Springer, 2006.

[AFK+08] J.-P. Aumasson, S. Fischer, S. Khazaei, W. Meier, and C. Rechberger. *New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba*. In K. Nyberg (ed.), FSE, volume 5086 of Lecture Notes in Computer Science, pages 470–488. Springer, 2008.

[AK03] F. Armknecht and M. Krause. *Algebraic Attacks on Combiners with Memory*. In Boneh [Bon03], pages 162–175.

[ALP04] F. Armknecht, J. Lano, and B. Preneel. *Extending the Resynchronization Attack*. In H. Handschuh and M. A. Hasan (eds.), Selected Areas in Cryptography, volume 3357 of Lecture Notes in Computer Science, pages 19–38. Springer, 2004.

[Arm04] F. Armknecht. *Improving Fast Algebraic Attacks*. In B. K. Roy and W. Meier (eds.), FSE, volume 3017 of Lecture Notes in Computer Science, pages 65–82. Springer, 2004.

[Arm06] F. Armknecht. Algebraic Attacks on Certain Stream Ciphers. PhD thesis, University of Mannheim, 2006.

[Bab95] S. Babbage. *A Space/Time Trade-Off in Exhaustive Search Attacks on Stream Ciphers*. In European Convention on Security and Detection, number 408. IEEE Conference Publication, 1995.

[BB05] E. Barkan and E. Biham. *Conditional Estimators: An Effective Attack on A5/1*. In Preneel and Tavares [PT06], pages 1–19.

[Bel11] S. M. Bellovin. *Frank Miller: Inventor of the One-Time Pad*. Cryptologia, 35(3):203–222, 2011.

[Ber05] D. J. Bernstein. *Salsa20*. Report 2005/025, eSTREAM, ECRYPT Stream Cipher Project, Apr. 2005. Retrieved Oct. 23, 2011 from: `http://www.ecrypt.eu.org/stream/p3ciphers/salsa20/salsa20_p3.zip`.

[Ber08] D. J. Bernstein. *Response to "Slid Pairs in Salsa20 and Trivium"*. pdf file released on `cr.yp.to`, 2008. Retrieved Oct. 23, 2011 from: `http://cr.yp.to/snuffle/reslid-20080925.pdf`.

[Ber11] D. J. Bernstein. *Extending the Salsa20 nonce*. In Symmetric Key Encryption Workshop (SKEW), Feb. 2011. Retrieved Oct. 23, 2011 from: `http://skew2011.mat.dtu.dk/proceedings/Extending%20the%20Salsa20%20nonce.pdf`.

[BG05] O. Billet and H. Gilbert. *Resistance of SNOW 2.0 Against Algebraic Attacks*. In A. Menezes (ed.), CT-RSA, volume 3376 of Lecture Notes in Computer Science, pages 19–28. Springer, 2005.

[BGW99] M. Briceno, I. Goldberg, and D. Wagner. *A Pedagogical Implementation of A5/1*, 1999. Retrieved Oct. 23, 2011 from: `http://cryptome.org/jya/a51-pi.htm`.

[BHNS10] B. B. Brumley, R. M. Hakala, K. Nyberg, and S. Sovio. *Consecutive S-box Lookups: A Timing Attack on SNOW 3G*. In M. Soriano, S. Qing, and J. López (eds.), ICICS, volume 6476 of Lecture Notes in Computer Science, pages 171–185. Springer, 2010.

[BL05] A. Braeken and J. Lano. *On the (Im)Possibility of Practical and Secure Nonlinear Filters and Combiners*. In Preneel and Tavares [PT06], pages 159–174.

[Bon03] D. Boneh (ed.). Advances in Cryptology – CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings, volume 2729 of Lecture Notes in Computer Science. Springer, 2003.

[BPR11] A. Bogdanov, B. Preneel, and V. Rijmen. *Security Evaluation of the K2 Stream Cipher*, Mar. 2011. Retrieved Oct. 23, 2011 from: `http://www.cryptrec.go.jp/estimation/techrep_id2010_2.pdf`.

[BPSZ10] A. Biryukov, D. Priemuth-Schmid, and B. Zhang. *Analysis of SNOW 3G; Resynchronization Mechanism*. In S. K. Katsikas and P. Samarati (eds.), SECRYPT, pages 327–333. SciTePress, 2010.

[BS90] E. Biham and A. Shamir. *Differential Cryptanalysis of DES-like Cryptosystems*. In A. Menezes and S. A. Vanstone (eds.),

CRYPTO, volume 537 of Lecture Notes in Computer Science, pages 2–21. Springer, 1990.

[BS00] A. Biryukov and A. Shamir. *Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers*. In T. Okamoto (ed.), ASIACRYPT, volume 1976 of Lecture Notes in Computer Science, pages 1–13. Springer, 2000.

[Buc65] B. Buchberger. Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. PhD thesis, University of Innsbruck, 1965.

[BW99] A. Biryukov and D. Wagner. *Slide Attacks*. In L. R. Knudsen (ed.), FSE, volume 1636 of Lecture Notes in Computer Science, pages 245–259. Springer, 1999.

[BW00] A. Biryukov and D. Wagner. *Advanced Slide Attacks*. In Preneel [Pre00], pages 589–606.

[Car06] C. Carlet. *On the Higher Order Nonlinearities of Algebraic Immune Functions*. In C. Dwork (ed.), CRYPTO, volume 4117 of Lecture Notes in Computer Science, pages 584–601. Springer, 2006.

[CF08] C. Carlet and K. Feng. *An Infinite Class of Balanced Functions with Optimal Algebraic Immunity, Good Immunity to Fast Algebraic Attacks and Good Nonlinearity*. In J. Pieprzyk (ed.), ASIACRYPT, volume 5350 of Lecture Notes in Computer Science, pages 425–440. Springer, 2008.

[CKM93] D. Coppersmith, H. Krawczyk, and Y. Mansour. *The Shrinking Generator*. In D. R. Stinson (ed.), CRYPTO, volume 773 of Lecture Notes in Computer Science, pages 22–39. Springer, 1993.

[CKPS00] N. Courtois, A. Klimov, J. Patarin, and A. Shamir. *Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations*. In Preneel [Pre00], pages 392–407.

[CM03] N. Courtois and W. Meier. *Algebraic Attacks on Stream Ciphers with Linear Feedback*. In E. Biham (ed.), EUROCRYPT, volume 2656 of Lecture Notes in Computer Science, pages 345–359. Springer, 2003.

[Coh95] F. Cohen. *A Short History of Cryptography*, 1995. Retrieved Oct. 23, 2011 from: `http://all.net/edu/curr/ip/Chap2-1.html`.

[Cou02] N. Courtois. *Higher Order Correlation Attacks, XL Algorithm and Cryptanalysis of Toyocrypt*. In P. J. Lee and C. H. Lim (eds.), ICISC, volume 2587 of Lecture Notes in Computer Science, pages 182–199. Springer, 2002.

[Cou03] N. Courtois. *Fast Algebraic Attacks on Stream Ciphers with Linear Feedback.* In Boneh [Bon03], pages 176–194.

[CP02] N. Courtois and J. Pieprzyk. *Cryptanalysis of Block Ciphers with Overdefined Systems of Equations.* In Y. Zheng (ed.), ASIACRYPT, volume 2501 of Lecture Notes in Computer Science, pages 267–287. Springer, 2002.

[Cro06] P. Crowley. *Truncated differential cryptanalysis of five rounds of Salsa20*, 2006. Retrieved Oct. 23, 2011 from: `http://www.ecrypt.eu.org/stream/papersdir/073.pdf`.

[CS91] V. V. Chepyzhov and B. J. M. Smeets. *On A Fast Correlation Attack on Certain Stream Ciphers.* In EUROCRYPT, pages 176–185, 1991.

[CYC06] CYCOM Cypher Research Laboratories. *A Brief History of Cryptography.* Last update: January 24, 2006. Retrieved Oct. 23, 2011 from: `http://www.cypher.com.au/crypto_history.htm`.

[DC09] B. Debraize and I. M. Corbella. *Fault Analysis of the Stream Cipher Snow 3G.* In L. Breveglieri, I. Koren, D. Naccache, E. Oswald, and J.-P. Seifert (eds.), FDTC, pages 103–110. IEEE Computer Society, 2009.

[dCP05] C. de Cannière and B. Preneel. *Trivium – A Stream Cipher Construction Inspired by Block Cipher Design Principles.* Report 2005/030, eSTREAM, ECRYPT Stream Cipher Project, Apr. 2005. Retrieved Oct. 23, 2011 from: `http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf`.

[DGM05] D. K. Dalai, K. C. Gupta, and S. Maitra. *Cryptographically Significant Boolean Functions: Construction and Analysis in Terms of Algebraic Immunity.* In H. Gilbert and H. Handschuh (eds.), FSE, volume 3557 of Lecture Notes in Computer Science, pages 98–111. Springer, 2005.

[DH76] W. Diffie and M. E. Hellman. *New Directions in Cryptography.* IEEE Transactions on Information Theory, 22(6):644–654, 1976.

[DR02] J. Daemen and V. Rijmen. The Design of Rijndael: AES – The Advanced Encryption Standard. Springer, 2002.

[DS09] I. Dinur and A. Shamir. *Cube Attacks on Tweakable Black Box Polynomials.* In A. Joux (ed.), EUROCRYPT, volume 5479 of Lecture Notes in Computer Science, pages 278–299. Springer, 2009.

[DS11] I. Dinur and A. Shamir. *Breaking Grain-128 with Dynamic Cube Attacks*. In A. Joux (ed.), FSE, volume 6733 of Lecture Notes in Computer Science, pages 167–187. Springer, 2011.

[ECR08] *ECRYPT: Network of Excellence in Cryptology*, 2004-2008. Retrieved Oct. 23, 2011 from: `http://www.ecrypt.eu.org/ecrypt1/`.

[EJ02] P. Ekdahl and T. Johansson. *A New Version of the Stream Cipher SNOW*. In K. Nyberg and H. M. Heys (eds.), Selected Areas in Cryptography, volume 2595 of Lecture Notes in Computer Science, pages 47–61. Springer, 2002.

[EJ03] P. Ekdahl and T. Johansson. *Another Attack on A5/1*. IEEE Transactions on Information Theory, 49(1):284–289, 2003.

[Ekd03] P. Ekdahl. On LFSR based Stream Ciphers – Analysis and Design. PhD thesis, Lund University, 2003.

[eST08] *eSTREAM: The ECRYPT Stream Cipher Project*, 2004-2008. Retrieved Oct. 23, 2011 from: `http://www.ecrypt.eu.org/stream/`.

[ETS06a] ETSI/SAGE. *Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2*. Document 2: SNOW 3G Specification, version 1.1, Sept. 2006. Retrieved Oct. 23, 2011 from: `http://cryptome.org/uea2-uia2/snow_3g_spec.pdf`.

[ETS06b] ETSI/SAGE. *Specification of the 3GPP Confidentiality and Integrity Algorithms UEA2 & UIA2*. Document 5: Design and Evaluation Report, version 1.1, Sept. 2006. Retrieved Oct. 23, 2011 from: `http://cryptome.org/uea2-uia2/uea2_design_evaluation.pdf`.

[Fau99] J.-C. Faugère. *A new efficient algorithm for computing Gröbner basis (F_4)*. Journal of Pure and Applied Algebra, 139:61–88, June 1999.

[Fau02] J.-C. Faugère. *A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5)*. pages 75–83, New York, NY, USA, 2002. ACM Press.

[Fis08] S. Fischer. Analysis of Lightweight Stream Ciphers. PhD thesis, École Polytechnique Fédérale de Lausanne, 2008.

[FKM08] S. Fischer, S. Khazaei, and W. Meier. *Chosen IV Statistical Analysis for Key Recovery Attacks on Stream Ciphers*. In S. Vaudenay (ed.), AFRICACRYPT, volume 5023 of Lecture Notes in Computer Science, pages 236–245. Springer, 2008.

[FM07] S. Fischer and W. Meier. *Algebraic Immunity of S-Boxes and Augmented Functions.* In A. Biryukov (ed.), FSE, volume 4593 of Lecture Notes in Computer Science, pages 366–381. Springer, 2007.

[FMB+06] S. Fischer, W. Meier, C. Berbain, J.-F. Biasse, and M. J. B. Robshaw. *Non-randomness in eSTREAM Candidates Salsa20 and TSC-4.* In R. Barua and T. Lange (eds.), INDOCRYPT, volume 4329 of Lecture Notes in Computer Science, pages 2–16. Springer, 2006.

[Gol97] J. D. Golic. *Cryptanalysis of Alleged A5 Stream Cipher.* In Advances in Cryptology – EUROCRYPT'97, volume 1233 of Lecture Notes in Computer Science, pages 239–255. Springer-Verlag, 1997.

[Hel80] M. E. Hellman. *A Cryptanalytic Time-Memory Trade-Off.* IEEE Transactions on Information Theory, 26(4):401–406, July 1980.

[Hel07] M. Hell. On the Design and Analysis of Stream Ciphers. PhD thesis, Lund University, 2007.

[Hon05] J. Hong. *certain pairs of key-IV pairs for Trivium.* ECRYPT Discussion Forum, article created at 05:11PM on September 13, 2005. Retrieved Oct. 23, 2011 from: `http://www.ecrypt.eu.org/stream/phorum/read.php?1,152`.

[HYY+10] M. Henricksen, W.-S. Yap, C. H. Yian, S. Kiyomoto, and T. Tanaka. *Side-Channel Analysis of the K2 Stream Cipher.* In R. Steinfeld and P. Hawkes (eds.), ACISP, volume 6168 of Lecture Notes in Computer Science, pages 53–73. Springer, 2010.

[JJ99] T. Johansson and F. Jönsson. *Fast Correlation Attacks Based on Turbo Code Techniques.* In Wiener [Wie99], pages 181–197.

[JJ00] T. Johansson and F. Jönsson. *Fast Correlation Attacks through Reconstruction of Linear Polynomials.* In M. Bellare (ed.), CRYPTO, volume 1880 of Lecture Notes in Computer Science, pages 300–315. Springer, 2000.

[Joh09] T. R. Johnson. *American Cryptology during the Cold War, 1945-1989. Book III: Retrenchment and Reform, 1972-1980, page 232.* File released on September 13, 2009. Retrieved Oct. 23, 2011 from: `http://cryptome.org/0001/nsa-meyer.htm`.

[Kah67a] D. Kahn. The Codebreakers – The Story of Secret Writing. Macmillan Publishing, first edition, 1967.

[Kah67b] D. Kahn. The Codebreakers – The Story of Secret Writing. Macmillan Publishing, first edition, 1967.

[Kas63] F. W. Kasiski. Die Geheimschriften und die Dechiffrir-Kunst (sic.). E. S. Mittler und Sohn, Berlin, 1863.

[Kel98] T. Kelly. *The Myth of the Skytale.* Cryptologia, 22(3):244–260, 1998.

[Ker83] A. Kerckhoffs. *La cryptographie militaire.* Journal des sciences militaires, IX:5–83, Jan. 1883.

[KJJ99] P. C. Kocher, J. Jaffe, and B. Jun. *Differential Power Analysis.* In Wiener [Wie99], pages 388–397.

[Koc96] P. C. Kocher. *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems.* In N. Koblitz (ed.), CRYPTO, volume 1109 of Lecture Notes in Computer Science, pages 104–113. Springer, 1996.

[KS99] A. Kipnis and A. Shamir. *Cryptanalysis of the HFE Public Key Cryptosystem by Relinearization.* In Wiener [Wie99], pages 19–30.

[KTS07] S. Kiyomoto, T. Tanaka, and K. Sakurai. *K2: A Stream Cipher Algorithm using Dynamic Feedback Control.* In J. Hernando, E. Fernández-Medina, and M. Malek (eds.), SECRYPT, pages 204–213. INSTICC Press, 2007.

[Lan06] J. Lano. Cryptanalysis and Design of Synchronous Stream Ciphers. PhD thesis, Katholieke Universiteit Leuven, 2006.

[LN97] R. Lidl and H. Niederreiter. Finite Fields, volume 20 of Encyclopedia of Mathematics and its Applications. Cambridge University Press, second edition, 1997.

[Mah45] A. P. Mahon. *The History of Hut Eight 1939 - 1945*, 1945. Retrieved Oct. 23, 2011 from: `http://www.ellsbury.com/hut8/hut8-000.htm`.

[Mas69] J. L. Massey. *Shift-Register Synthesis and BCH Decoding.* IEEE Transactions on Information Theory, 15(1):122–127, 1969.

[Max06] A. Maximov. Some Words on Cryptanalysis of Stream Ciphers. PhD thesis, Lund University, 2006.

[MB07] A. Maximov and A. Biryukov. *Two Trivial Attacks on Trivium.* In C. M. Adams, A. Miri, and M. J. Wiener (eds.), Selected Areas in Cryptography, volume 4876 of Lecture Notes in Computer Science, pages 36–55. Springer, 2007.

[MCP07] C. McDonald, C. Charnes, and J. Pieprzyk. *Attacking Bivium with MiniSat*. Report 2007/040, eSTREAM, ECRYPT Stream Cipher Project, Apr. 2007. Retrieved Oct. 23, 2011 from: `http://www.ecrypt.eu.org/stream/papersdir/2007/040.pdf`.

[MG90] M. J. Mihaljevic and J. D. Golic. *A Fast Iterative Algorithm For A Shift Register Initial State Reconstruction Given The Noisy Output Sequence*. In J. Seberry and J. Pieprzyk (eds.), AUSCRYPT, volume 453 of Lecture Notes in Computer Science, pages 165–175. Springer, 1990.

[Mil01] A. R. Miller. *The Cryptographic Mathematics of Enigma*. Technical report, National Security Agency, 2001. Retrieved Oct. 23, 2011 from: `http://www.nsa.gov/about/_files/cryptologic_heritage/publications/wwii/engima_cryptographic_mathematics.pdf`.

[MPC04] W. Meier, E. Pasalic, and C. Carlet. *Algebraic Attacks and Decomposition of Boolean Functions*. In C. Cachin and J. Camenisch (eds.), EUROCRYPT, volume 3027 of Lecture Notes in Computer Science, pages 474–491. Springer, 2004.

[MS89] W. Meier and O. Staffelbach. *Fast Correlation Attacks on Certain Stream Ciphers*. Journal of Cryptology, 1(3):159–176, 1989.

[MS94] W. Meier and O. Staffelbach. *The Self-Shrinking Generator*. In A. D. Santis (ed.), EUROCRYPT, volume 950 of Lecture Notes in Computer Science, pages 205–214. Springer, 1994.

[MvOV96] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. Handbook of Applied Cryptography. Publisher CRC Press, first edition, 1996.

[NTW10] K. Nohl, E. Tews, and R.-P. Weinmann. *Cryptanalysis of the DECT Standard Cipher*. In S. Hong and T. Iwata (eds.), FSE, volume 6147 of Lecture Notes in Computer Science, pages 1–18. Springer, 2010.

[NW06] K. Nyberg and J. Wallén. *Improved Linear Distinguishers for SNOW 2.0*. In M. J. B. Robshaw (ed.), FSE, volume 4047 of Lecture Notes in Computer Science, pages 144–162. Springer, 2006.

[Pen96] W. T. Penzhorn. *Correlation Attacks on Stream Ciphers: Computing Low-Weight Parity Checks Based on Error-Correcting Codes*. In D. Gollmann (ed.), FSE, volume 1039 of Lecture Notes in Computer Science, pages 159–172. Springer, 1996.

[Pre00] B. Preneel (ed.). Advances in Cryptology – EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding, volume 1807 of Lecture Notes in Computer Science. Springer, 2000.

[PS11] D. Priemuth-Schmid. *Attacks on Simplified Versions of K2.* In P. Bouvry, M. A. Kłopotek, F. Leprevost, M. Marciniak, A. Mykowiecka, and H. Rybiński (eds.), SIIS, volume 7053 of Lecture Notes in Computer Science, pages 117–127. Springer, 2011.

[PSB08] D. Priemuth-Schmid and A. Biryukov. *Slid Pairs in Salsa20 and Trivium.* In D. R. Chowdhury, V. Rijmen, and A. Das (eds.), INDOCRYPT, volume 5365 of Lecture Notes in Computer Science, pages 1–14. Springer, 2008.

[PT06] B. Preneel and S. E. Tavares (eds.). Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers, volume 3897 of Lecture Notes in Computer Science. Springer, 2006.

[Rad06] H. Raddum. *Cryptanalytic Results on Trivium.* Report 2006/039, eSTREAM, ECRYPT Stream Cipher Project, Mar. 2006. Retrieved Oct. 23, 2011 from: `http://www.ecrypt.eu.org/stream/papersdir/2006/039.ps`.

[Rec04] C. Rechberger. *Side-Channel Analysis of Stream Ciphers*, 2004.

[RSA78] R. L. Rivest, A. Shamir, and L. M. Adleman. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM, 21(2):120–126, 1978.

[RSA83] R. L. Rivest, A. Shamir, and L. M. Adleman. *Cryptographic Communications System and Method.* U.S. Patent 4,405,829, 1983. Retrieved Oct. 23, 2011 from: `http://www.google.com/patents?vid=4405829`.

[Sha45] C. E. Shannon. *A Mathematical Theory of Cryptography.* Memorandum MM 45-110-02, Bell Laboratories, 114 pp. + 25 figs. Classified report, superseded by the following paper, Sept. 1945.

[Sha48] C. E. Shannon. *A Mathematical Theory of Communication.* Bell System Technical Journal, 27:379–423 and 623–656, July and Oct. 1948.

[Sha49] C. E. Shannon. *Communication Theory of Secrecy Systems.* Bell System Technical Journal, 28(4):656–715, 1949.

[Sie84] T. Siegenthaler. *Correlation-Immunity of Nonlinear Combining Functions for Cryptographic Applications.* IEEE Transactions on Information Theory, 30(5):776–780, 1984.

[Sie85] T. Siegenthaler. *Decrypting a Class of Stream Ciphers Using Ciphertext Only.* IEEE Transactions on Computers, 34(1):81–85, 1985.

[Sin99] S. Singh. The Code Book. Fourth Estate, first edition, 1999.

[TK07] M. S. Turan and O. Kara. *Linear Approximations for 2-round Trivium.* In SASC 2007 – The State of the Art of Stream Ciphers, 2007. Retrieved Oct. 23, 2011 from: `http://www.ecrypt.eu.org/stream/papersdir/2007/008.pdf`.

[TSK+07] Y. Tsunoo, T. Saito, H. Kubo, T. Suzaki, and H. Nakashima. *Differential Cryptanalysis of Salsa20/8.* In SASC 2007 – The State of the Art of Stream Ciphers, 2007. Retrieved Oct. 23, 2011 from: `http://www.ecrypt.eu.org/stream/papersdir/2007/010.pdf`.

[Ver19] G. S. Vernam. *SECRET SIGNALING SYSTEM.* U.S. Patent 1,310,719, 1919. Retrieved Oct. 23, 2011 from: `http://www.google.com/patents?vid=1310719`.

[Ver26] G. S. Vernam. *Cipher Printing Telegraph Systems For Secret Wire and Radio Telegraphic Communications.* Journal of American Institute of Electrical Engineers, 45:109–115, 1926.

[Vie07] M. Vielhaber. *Breaking ONE.Fivium by AIDA an Algebraic IV Differential Attack.* Report 2007/413, Cryptology ePrint Archive, Oct. 2007. Retrieved Oct. 23, 2011 from: `http://eprint.iacr.org/2007/413.pdf`.

[WBdC03] D. Watanabe, A. Biryukov, and C. de Cannière. *A Distinguishing Attack of SNOW 2.0 with Linear Masking Method.* In M. Matsui and R. J. Zuccherato (eds.), Selected Areas in Cryptography, volume 3006 of Lecture Notes in Computer Science, pages 222–233. Springer, 2003.

[Wie99] M. J. Wiener (ed.). Advances in Cryptology – CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings, volume 1666 of Lecture Notes in Computer Science. Springer, 1999.

[Wik05] Wikimedia Commons. *Enigma Machine at the Imperial War Museum, London.* Image photographed and published by Karsten Sperling, 2005. Retrieved Oct. 23, 2011 from: `http://commons.wikimedia.org/wiki/File:EnigmaMachineLabeled.jpg`.

[Wik07] Wikimedia Commons. *Skytale*. Image published by user Luringen, 2007. Retrieved Oct. 23, 2011 from: `http://commons.wikimedia.org/wiki/File:Skytale.png`.

[Wik11] Wikipedia, The Free Encyclopedia. *English and German texts concerning miscellaneous topics*, 2011. Available Oct. 23, 2011 at: `http://en.wikipedia.org/wiki/Main_Page` and `http://de.wikipedia.org/wiki/Hauptseite`.

[WSLM05] D. Whiting, B. Schneier, S. Lucks, and F. Muller. *Phelix – Fast Encryption and Authentication in a Single Cryptographic Primitive*. Report 2005/020, eSTREAM, ECRYPT Stream Cipher Project, Apr. 2005. Retrieved Oct. 23, 2011 from: `http://www.ecrypt.eu.org/stream/p2ciphers/phelix/phelix_p2.pdf`.

Appendix A

Salsa20 System of Equations for a Slid Pair

Word equations given by the equations $S' = F(S)$ and $X' = F(X)$.

$$
\begin{aligned}
k'_0 &= k_3 \oplus ((\sigma_0 + k_5) \lll 7) & x'_1 &= x_4 \oplus ((x_0 + x_{12}) \lll 7)\\
k'_1 &= c_0 \oplus ((k'_0 + \sigma_0) \lll 9) & x'_2 &= x_8 \oplus ((x'_1 + x_0) \lll 9)\\
k'_2 &= k_5 \oplus ((k'_1 + k'_0) \lll 13) & x'_3 &= x_{12} \oplus ((x'_2 + x'_1) \lll 13)\\
\sigma_0 &= \sigma_0 \oplus ((k'_2 + k'_1) \lll 18) & x'_0 &= x_0 \oplus ((x'_3 + x'_2) \lll 18)\\
n'_0 &= c_1 \oplus ((\sigma_1 + k_0) \lll 7) & x'_6 &= x_9 \oplus ((x_5 + x_1) \lll 7)\\
n'_1 &= k_6 \oplus ((n'_0 + \sigma_1) \lll 9) & x'_7 &= x_{13} \oplus ((x'_6 + x_5) \lll 9)\\
k'_3 &= k_0 \oplus ((n'_1 + n'_0) \lll 13) & x'_4 &= x_1 \oplus ((x'_7 + x'_6) \lll 13)\\
\sigma_1 &= \sigma_1 \oplus ((k'_3 + n'_1) \lll 18) & x'_5 &= x_5 \oplus ((x'_4 + x'_7) \lll 18)\\
k'_4 &= k_7 \oplus ((\sigma_2 + n_0) \lll 7) & x'_{11} &= x_{14} \oplus ((x_{10} + x_6) \lll 7)\\
c'_0 &= k_1 \oplus ((k'_4 + \sigma_2) \lll 9) & x'_8 &= x_2 \oplus ((x'_{11} + x_{10}) \lll 9)\\
c'_1 &= n_0 \oplus ((c'_0 + k'_4) \lll 13) & x'_9 &= x_6 \oplus ((x'_8 + x'_{11}) \lll 13)\\
\sigma_2 &= \sigma_2 \oplus ((c'_1 + c'_0) \lll 18) & x'_{10} &= x_{10} \oplus ((x'_9 + x'_8) \lll 18)\\
k'_5 &= k_2 \oplus ((\sigma_3 + k_4) \lll 7) & x'_{12} &= x_3 \oplus ((x_{15} + x_{11}) \lll 7)\\
k'_6 &= n_1 \oplus ((k'_5 + \sigma_3) \lll 9) & x'_{13} &= x_7 \oplus ((x'_{12} + x_{15}) \lll 9)\\
k'_7 &= k_4 \oplus ((k'_6 + k'_5) \lll 13) & x'_{14} &= x_{11} \oplus ((x'_{13} + x'_{12}) \lll 13)\\
\sigma_3 &= \sigma_3 \oplus ((k'_7 + k'_6) \lll 18) & x'_{15} &= x_{15} \oplus ((x'_{14} + x'_{13}) \lll 18)
\end{aligned}
$$

Word equations given by the feedforward for the first keystream words and for the second keystream words.

$$
\begin{aligned}
z_0 &= x_0 + \sigma_0 & z'_0 &= x'_0 + \sigma_0 \\
z_1 &= x_1 + k_0 & z'_1 &= x'_1 + k'_0 \\
z_2 &= x_2 + k_1 & z'_2 &= x'_2 + k'_1 \\
z_3 &= x_3 + k_2 & z'_3 &= x'_3 + k'_2 \\
z_4 &= x_4 + k_3 & z'_4 &= x'_4 + k'_3 \\
z_5 &= x_5 + \sigma_1 & z'_5 &= x'_5 + \sigma_1 \\
z_6 &= x_6 + n_0 & z'_6 &= x'_6 + n'_0 \\
z_7 &= x_7 + n_1 & z'_7 &= x'_7 + n'_1 \\
z_8 &= x_8 + c_0 & z'_8 &= x'_8 + c'_0 \\
z_9 &= x_9 + c_1 & z'_9 &= x'_9 + c'_1 \\
z_{10} &= x_{10} + \sigma_2 & z'_{10} &= x'_{10} + \sigma_2 \\
z_{11} &= x_{11} + k_4 & z'_{11} &= x'_{11} + k'_4 \\
z_{12} &= x_{12} + k_5 & z'_{12} &= x'_{12} + k'_5 \\
z_{13} &= x_{13} + k_6 & z'_{13} &= x'_{13} + k'_6 \\
z_{14} &= x_{14} + k_7 & z'_{14} &= x'_{14} + k'_7 \\
z_{15} &= x_{15} + \sigma_3 & z'_{15} &= x'_{15} + \sigma_3
\end{aligned}
$$

Appendix B

SNOW 3G Simplify the Equation with two S-boxes

We want to simplify the equation

$$\overset{\text{out}}{\Delta} S_2(\Delta k_1) \oplus \overset{\text{out}}{\Delta} S_1(\Delta k_2) = \Delta k_4 \ .$$

The main difficulty is that S_1 and S_2 use the same MixColumns matrix but over two different fields $GF(2^8)$. We start with rewriting this equation in byte notation as

$$MC_2 \cdot \begin{pmatrix} \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(0)}) \\ \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(1)}) \\ \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(2)}) \\ \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(3)}) \end{pmatrix} \oplus MC_1 \cdot \begin{pmatrix} \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(0)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(1)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(2)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(3)}) \end{pmatrix} = \begin{pmatrix} \Delta k_4^{(0)} \\ \Delta k_4^{(1)} \\ \Delta k_4^{(2)} \\ \Delta k_4^{(3)} \end{pmatrix} \ .$$

Then multiplying this equation with the inverse matrix MC_1^{-1}, we get

$$MC_1^{-1} \cdot \left(MC_2 \cdot \begin{pmatrix} \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(0)}) \\ \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(1)}) \\ \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(2)}) \\ \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(3)}) \end{pmatrix} \right) \oplus \begin{pmatrix} \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(0)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(1)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(2)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(3)}) \end{pmatrix} = MC_1^{-1} \cdot \begin{pmatrix} \Delta k_4^{(0)} \\ \Delta k_4^{(1)} \\ \Delta k_4^{(2)} \\ \Delta k_4^{(3)} \end{pmatrix}.$$

If we expand the matrix multiplications and have a look at the byte vectors it shows that the first entry of the first vector contains the byte $\overset{\text{out}}{\Delta} S_Q(\Delta k_1^0)$ and a byte polynomial containing only the most significant bits of all four values $\overset{\text{out}}{\Delta} S_Q$. We denote this polynomial with p_0^{msb}. The other three rows have similar structures but with different polynomials p_i^{msb}

$(i = 1, 2, 3)$. Therefore, we can rewrite the equation to

$$\begin{pmatrix} \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(0)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(1)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(2)}) \\ \overset{\text{out}}{\Delta} S_R(\Delta k_2^{(3)}) \end{pmatrix} = \begin{pmatrix} \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(0)}) \\ \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(1)}) \\ \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(2)}) \\ \overset{\text{out}}{\Delta} S_Q(\Delta k_1^{(3)}) \end{pmatrix} \oplus \begin{pmatrix} p_0^{\text{msb}} \\ p_1^{\text{msb}} \\ p_2^{\text{msb}} \\ p_3^{\text{msb}} \end{pmatrix} \oplus MC_1^{-1} \cdot \begin{pmatrix} \Delta k_4^{(0)} \\ \Delta k_4^{(1)} \\ \Delta k_4^{(2)} \\ \Delta k_4^{(3)} \end{pmatrix}.$$

By m_0 we denote the most significant bit of the value $\overset{\text{out}}{\Delta} S_Q(\Delta k_1^0)$ and with m_1 the most significant bit of the value $\overset{\text{out}}{\Delta} S_Q(\Delta k_1^1)$ as well as m_2 for $\overset{\text{out}}{\Delta} S_Q(\Delta k_1^2)$ and m_3 for $\overset{\text{out}}{\Delta} S_Q(\Delta k_1^3)$. Then the polynomials p_i^{msb} $i = 0, \ldots, 3$ are

$$\begin{aligned}
p_0^{msb} &= (m_1 \oplus m_3)x^7 + (m_0 \oplus m_1)x^6 + (m_2 \oplus m_3)x^5 + (m_1 \oplus m_2)x^4 \\
&\quad + (m_0 \oplus m_2)x^2 + (m_1 \oplus m_2)x + (m_0 \oplus m_1 \oplus m_2 \oplus m_3) \\
p_1^{msb} &= (m_0 \oplus m_2)x^7 + (m_1 \oplus m_2)x^6 + (m_0 \oplus m_3)x^5 + (m_2 \oplus m_3)x^4 \\
&\quad + (m_1 \oplus m_3)x^2 + (m_2 \oplus m_3)x + (m_0 \oplus m_1 \oplus m_2 \oplus m_3) \\
p_2^{msb} &= (m_1 \oplus m_3)x^7 + (m_2 \oplus m_3)x^6 + (m_0 \oplus m_1)x^5 + (m_0 \oplus m_3)x^4 \\
&\quad + (m_0 \oplus m_2)x^2 + (m_0 \oplus m_3)x + (m_0 \oplus m_1 \oplus m_2 \oplus m_3) \\
p_3^{msb} &= (m_0 \oplus m_2)x^7 + (m_0 \oplus m_3)x^6 + (m_1 \oplus m_2)x^5 + (m_0 \oplus m_1)x^4 \\
&\quad + (m_1 \oplus m_3)x^2 + (m_0 \oplus m_1)x + (m_0 \oplus m_1 \oplus m_2 \oplus m_3) \ .
\end{aligned}$$